1971

A general problem describer for computer assisted instruction.

Wools, Ronald Joe

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/15732

A GENERAL PROBLEM DESCRIBER FOR COMPUTER ASSISTED INSTRUCTION

Ronald Joe Wools

United States
Naval Postgraduate School

THESIS

A GENERAL PROBLEM DESCRIBER
FOR
COMPUTER ASSISTED INSTRUCTION

by

Ronald Joe Wools

Thesis Advisor: R.C. Bolles

June 1971

A General Problem Describer for Computer Assisted Instruction

by

Ronald Joe Wools
Lieutenant, United States Navy
B.S., Indiana State University, 1963

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1971

ABSTRACT

Currently in computer assisted instruction systems a number of problems are presented to each student during a problem session and each individual problem is specified by the author of the session. A better approach might be to provide the author with a language in which he can describe the general type of problem he wants his students to be taught and let the machine generate the specific problems. This would relieve the teacher of the task of writing out a whole series of problems for each general concept he wishes to teach. This thesis presents a subset of English and mathematical notation which the teacher can use to describe a general problem type. The PROBLEM DESCRIPTION PROCESSOR accepts the general problem description and produces a low level language which is used by the PROBLEM DESCRIPTION INTERPRETER to produce specific problems. This system works for fourth grade arithmetic problems and could be extended for use in other areas of instruction.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

As the population of the United States increases and as technology and behavior become more complex, the need for quality education in schools and universities becomes more compelling. But the need is not limited to quality. The quantity of instruction is being hard pressed to maintain the same growth rate as enrollments. In California, the State Superintendent of Public Instruction, Wilson Riles, has stated that student enrollments are higher this school year than they were last year, but the state has 9000 fewer teachers. This means that other ways must be found to maintain quality in the United States.

Computer assisted instruction is one of the possible ways in which some of the critical areas of education may be improved. Some of the areas which need improvement are providing individualized instruction and relieving the teacher of administrative tasks. The research that has been done for this thesis and computer assisted instruction in general has only scratched the surface in trying to get rid of these problems.

Since the effectiveness of a learning task depends on the efficiency of communication it achieves, the focus of learning should be narrow; the learner should not be required to give significant attention or effort to behaviors irrelevant to the instructional goals; and the

quantum of learning should be kept small enough so that the learning situation can be properly and confidently evaluated by the learner. At the same time the learning situation should be rich enough to convey the exact characterization in minute detail.

Conventional instruction systems depend on humans to carry out the crucial functions of response evaluation and motivation of the student. Where learning tasks can be devised that use a computer terminal, interactive computer assisted instruction could, in principle, take over the functions of evaluation and motivation of students if a task evaluation algorithm and an algorithm of control can be made. Numerical evaluation systems are of special value and importance because they lend themselves to mathematical modeling.

This research is an attempt at the design and implementation of a very small section of computer assisted instruction which describe a general type of problem and specifies the constraints on each problem. The idea is to be able to use a more natural language to describe a general type of fourth grade arithmetic problem. It is felt that the more the problem description language approximates English the easier it would be for a non-programmer to use. The language presented in this thesis is still somewhat restricted because it consists of only a subset of English and has a somewhat set form. The overall system provides an interface between the computer and the teacher for the description, selection, and presentation of problems to students.

II. BACKGROUND

The field of instruction technology presents a rather complex picture. Any attempt at reviewing this new and rapidly changing field would be incomplete and approximate at best. An instructional system includes the following elements: learner, materials, monitor, author--teacher, and administrator. The picture becomes more complex with the addition of other students, teachers, teaching assistants, special projects, and outside reading. The dynamic interaction of materials, strategies, and communication channels over time must be considered in any complex analysis. There are a number of basic and distinct, though interdependent, aspects or functions of computer assisted systems.

A common view is that computer assisted instruction is an automated form of programmed instruction. It should be no surprise to find that many implementations of computer assisted instruction use this simplistic approach of automating the existing methods. Other "teaching machines" permit some changes in the format. But with the capabilities of computers and display devices which are available, one should not be constrained to variations of the earlier paper and pencil format. Instead of simply automating programmed instruction, new computer based instructional systems should fully exploit the capabilities of the computer. This is being investigated by this and other research at the Naval Postgraduate School.

In this section, various modes of computer assisted instruction will be discussed as well as what can, in general, be done with computers in education.

A. MODES OF COMPUTER ASSISTED INSTRUCTION

There are different areas in which computer assisted instruction can operate. These have been widely discussed in the literature. In this section five general types of computer assisted instruction systems are discussed: drill and practice, inquiry, tutorial, problem solving, and author.

1. Drill and Practice

The computer is used to present a sequence of spelling or arithmetic problems and the system allows the student numerous opportunities for response. The introduction of concepts and new ideas is handled in conventional fashion by the teacher. In the case of elementary mathematics each student would receive daily a certain number of exercises, which would be automatically presented, evaluated, and scored by the computer program without any interference by the teacher. These exercises can be presented on an individualized basis, with the brighter students receiving exercises that are harder than the average, and the slower students receiving easier problems. In using a computer in this way, it is not necessary to decide which category a student fits in at the beginning of the school year and a student does not need to stay in the same category for the entire year. In this context, category means both the grade level of the student and

the difficulty of problem he can work. Individualized drill and practice work is suitable to most of the elementary subjects which make up the curriculum. Typical parts of the curriculum which benefit from standardized and regular drill and practice exercises are elementary mathematics, elementary science, and the beginning work in foreign language.

2. Inquiry

This mode has often been called information retrieval and can easily be used by computer assisted instruction to answer general questions asked by a student. In this mode the student uses a natural language. He asks questions which are addressed to the system. The system typically processes the questions using key-words, hashing codes and search algorithms to retrieve an answer. Systems Development Corporation has a program called CONVERSE. This program has been designed to provide answers to questions posed in a limited subset of English. Using an existing data management system, CONVERSE translates an English question into one or more file-searching procedures. If complete translation is not possible, the program provides the user with information that may help him in rephrasing his questions into acceptable English terms.

A system has been built that attempts to answer students' questions from an encyclopedia which has been placed in the computer's memory. The first step in answering the question is to make a search for the smallest unit of text in the data base, preferably a sentence containing the

intersection of the greatest number of content words contained in the question. A simple information score is used to weight some words more heavily than others in selecting potential answers. The highest scoring answers are then retrieved from the tape on which the original text was stored, for the students' answer. The approach is to successively filter out more and more irrelevant information, leaving only statements which have the highest probability of being answers to the question.

3. Tutorial

This is a level of instruction that not only involves questions and answers between the computer and students during the presentation of instructional material, but may involve other modes of computer assisted instruction. The other modes become classes of instructional experience that can comprise a more general educational experience for the students. The computer program takes over the responsibility both for presenting a concept and for developing skill in its use. The intention is to approximate the interaction between a patient tutor and an individual student. The program has to have enough flexibility to ensure that the bright students do not become bored with endless repetitive exercises and ensure that the slower students do not become discouraged with initial failures. As soon as the student has a clear understanding of a concept on the basis of his handling of a number of exercises, he is moved on to a

new concept and new exercises. The formulation of teaching strategies is the biggest problem facing the authors of this mode.

4. Problem Solving

Problem solving can be looked upon in two different ways. The first is the use of a computer to solve quantitative problems and the student uses a language like FORTRAN or BASIC to accomplish his purpose. He writes a program and enters his data. But this is a narrow approach when looking at the second method which is a considerably more complex system involving the modelling of human problem solving capability. This type of system is best represented by the Newell, Shaw, and Simon work in 1959 on General Problem Solver. A set of complex means-ends analysis routines make up the main part of the program.

5. Author

This is the most primitive in the state of development. The system's capability depends upon its ability to generate specified kinds of materials, to load them on the system, to select the best modes of computer assisted instruction, and to present the material in modules which minimize the teaching-learning effort. Within instructional modes, a number of variations are still possible, so an algorithm must be used to select not only the mode of instruction but also particular variations to use with it. To locate within a mode the particular variation that is wanted, there have to be contingency rules that depend upon who the student is

and how he has performed. This differs from the tutorial mode in that the computer in conjunction with the student decides what is to be presented.

B. COMPUTERS IN EDUCATION

The nature of a learning activity may be more or less artificial, depending upon the extent to which the behavior being developed is a subskill of the final behavior or a rough approximation to it. In any case the learning task itself has content and involves behaviors that are irrelevant to the final instructional goals, but are necessary for purposes of doing the task; examples of such secondary skills are the use of a keyboard, the format rules of messages, the procedures used in a test administration. Computer assisted instruction systems should be carefully designed to provide flexible, interactive capabilities without introducing too much of this type of overhead.

Recently elementary methematics has been reorganized around the idea of presenting fundamental mathmatical concepts which logically build upon each other. For example, the concept of a set is presented, addition is described in terms of sets, etc. This approach lends itself very nicely to computer directed tutoring because new concepts and tutoring hints can be based upon more primitive concepts.

A computer-based system allows interim and sometimes automatic decisions in the course of a session. In addition to reducing the total elapsed time required for execution

of a session, the computer can keep a record of the full interaction with the student with three major gains. The record of performance for the teacher allows easy and pin-pointed experimentation with alternate ways of presenting material of various sorts to various students, thus improving understanding and practice in education. A record of what the student has learned and his particular learning features can govern the rules of thumb used by the tutoring program. This record shows the teaching methods that are the best for the individual student. It would show such things as: whether ideas are best presented first by example or introduced initially as general principles, whether small steps and repetition or great mental strides are needed, or whether visual or auditory presentation is most helpful. The third gain of the performance record is a way of determining if a student has in fact demonstrated a certain level of comprehension. The computer can teach and independently certify achievement.

III. DESIGN OF THE PROBLEM DESCRIBER

The concepts which every computer assisted instruction system should use are the ideas of students moving at their own pace, individualized instruction, and a more flexible school system. If these ideas are used in conjunction with a computer system which has complete graphics terminals, they could enhance a number of areas such as arithmetic, geometry,

set theory, algebra, etc. This thesis concerns itself mostly with fourth grade mathematics, but the ideas could be easily extended to cover these other areas.

In looking at computer assisted instruction there are various routes which can be taken in the development of a tutorial system. One of these routes and the most basic is to look at one specific problem at a time. The teacher would tell the computer the exact problem that he wants. A sample set of addition problems with no carries would be:

$$(1)\quad \begin{array}{r} 27 \\ +52 \\ \hline \end{array} \quad \begin{array}{r} 31 \\ +45 \\ \hline \end{array} \qquad \begin{array}{r} 72 \\ +17 \\ \hline \end{array} \quad \begin{array}{r} 51 \\ +38 \\ \hline \end{array}$$

This method requires the teacher to make up a series of problems for each concept to be covered. Another disadvantage of this method is that it does not lend itself to much variety of problem presentation because each student works the same set of problems. This is essentially the same thing as writing out a set of problems on a spirit master, having it reproduced, and handing it out to the students to be worked. Without variety the bright students become bored with much repetition and the slower students will easily become discouraged when they can not answer some of the questions.

If a teacher had a language available which provided him with a method of describing general types of problems he could describe a problem type and let the machine produce specific problems which demonstrate the desired concept. In this way a terminal session could be designed to generate a variable

number of problems depending upon how quickly the student learned the material. Each time a problem of a given type is desired a reference is made to the proper problem description and the machine calls on a random number generator to produce a specific problem. Notice that, since a random element is built into this approach, different students will see different problems - all of which are examples demonstrating the concept included in the problem description. The teacher describes the type of problem once and refers to this description as often as he likes.

There are a number of different ways a language could be constructed to describe a type of problem. The rest of this section will discuss a series of four possible languages: (1) a low level language involving detailed specifications for each digit; (2) a higher level language which allows some mathematical notation; (3) a higher level language which allows a subset of English and mathematical notation; and (4) full English. Two example specifications of problem types will be used in this discussion: (1) an addition problem without carries; and (2) an addition involving one carry in the tens column.

The low level language requires that the number of digits and the limits on each digit be explicitly stated. For example the addition problem without carries would be described as follows:

```
(2)  X4X3X2X1 + Y3Y2Y1 = answer
     X4 = RANDOM DIGIT (0 to 9)
     X3 = RANDOM DIIIT (0 to 9)
     X2 = RANDOM DIGIT (0 to 9)
     X1 - RANDOM DIGIT (0 to 9)
     Y3 = RANDOM DIGIT (0 to (9 - value of X3)
     Y2 = RANDOM DIGIT (0 to (9 - value of X2)
     Y1 = RANDOM DIGIT (0 to (9 - value of X1)
```

The first line indicates that the problem is an addition problem involving a 4-digit number and a 3-digit number. The remaining lines explicitly state the procedure to be used to generate values for these digits. Specific problems which the computer might generate for this description might be:

```
2791     5411
+103     +536
```

The addition problem which includes a carry in the tens column would be described as follows:

```
(3)   X4X3X2X1 + Y3Y2Y1 = Answer
      X4 - RANDOM DIGIT (0 to 9)
      X3 - RANDOM DIGIT (0 to 9)
      X2 = RANDOM DIGIT (0 to 9)
      X1 = RANDOM DIGIT (0 to 9)
      Y3 = RANDOM DIGIT (0 to (8 - value of X3)
      Y2 = RANDOM DIGIT ((10 - value of X2) to 9)
      Y1 = RANDOM DIGIT (0 to (9 - value of X1)
```

Specific problems might be:

```
8547     2563
+271     +156
```

Looking at a higher approach, why not describe the general problem and the limits on the digits in mathematical notation. This will add more flexibility and make things a little easier for the teacher. The computer program would be more involved, but it could still generate problems of

the type specified. Using mathematical notation the addition problem with no carries would look like this:

```
(4)  X4X3X2X1 + Y3Y2Y1 = Answer
     Y3 <= 9 - X3
     Y2 <= 9 - X2
     Y1 <= 9 - X1
```

The problem which has carries in the tens position would be:

```
(5)  X4X3X2X1 + Y3Y2Y1 = Answer
     X2 >= 1
     Y3 <= 8 - X3
     Y2 <= 10 - X2
     Y1 <= 9 - X1
```

Notice that this method should be relatively easy for the elementary teacher to use, but it is still not quite the way one teacher would describe a general type of problem to another teacher.

By using a small subset of informal English with mathematical notation, the concept of general problem description can be broadened even more. What English phrases are needed to be able to state a general problem? Typical phrases are "carries in tens place," "borrows in tens place," and "carries in position two." In order to specify the direction the problem is to be worked, "vertical problem direction," "vertical," "hor," or something along these lines is needed. With this additional flexibility the first example could be specified as follows:

```
(6)  X4X3X2X1 + Y3Y2Y1 = Answer
     No carries with a vertical problem direction.
```

or

```
(7)  X4X3X2X1 + Y3Y2Y1 = Answer
     Vert no carries.
```

The second example could be:

```
    (8)  X4X3X2X1 + Y3Y2Y1 = Answer
         Carries in tens place hor.
```

A problem with complex constraints may be written as:

```
    (9)  X4X3X2X1 + Y3Y2Y1 = Answer
         Horizontal
         X4 > X3
         X3 > X2
         X2 > X1
         Y3 < 9 - X3
         Y2 ≠ X2
```

The highest level of problem description which can be achieved is to be able to have the full use of English. This gives the average elementary teacher who is a non-programmer of computers a more flexible and understanding way of describing problems. A problem description including administrative information, tutoring hints, etc., might be described in English as:

(10) Generate ten addition problems with carries in tens place. First, show the student two examples. If there are any errors in the student's work, demonstrate how the problem should have been worked.

Or another problem description might be:

(11) Demonstrate two addition problems with carries in tens place presenting the problem horizontally then generate ten of the same type of problems for a quiz.

This method is only possible when a language processor is developed which will accept valid English sentences and convert these sentences into the low level language.

IV. IMPLEMENTATION OF THE PRCBLEM DESCRIBER

At the beginning of the implementation phase, a decision had to be made as to which language to use. There are many languages available but the one chosen for this work in the construction of a general problem describer was XPL. One of the main reasons XPL was chosen is that it has a syntax directed skeleton compiler with an associated syntax analyzer. The syntax analyzer is used to construct tables which this skeleton compiler uses to make reduction decisions during compilation.

The most desirable way to describe a general problem is one which has the full use of English. For practical considerations, the language developed in this thesis uses a subset of English and mathematical notation (i.e., the third language mentioned in Section III). In order to implement a system like this the following is needed: (1) an interpreter that accepts a language based on the low level language in Section III; and (2) a language processor which changes the subset of English and mathematical notation into the low level language. The flow of information which is accomplished by this system is shown in Figure 1.

A. PROBLEM DESCRIPTION INTERPRETER

The PROBLEM DESCRIPTION INTERPRETER interprets the description of problem type written in the low level language and produces a specific problem fitting the description.

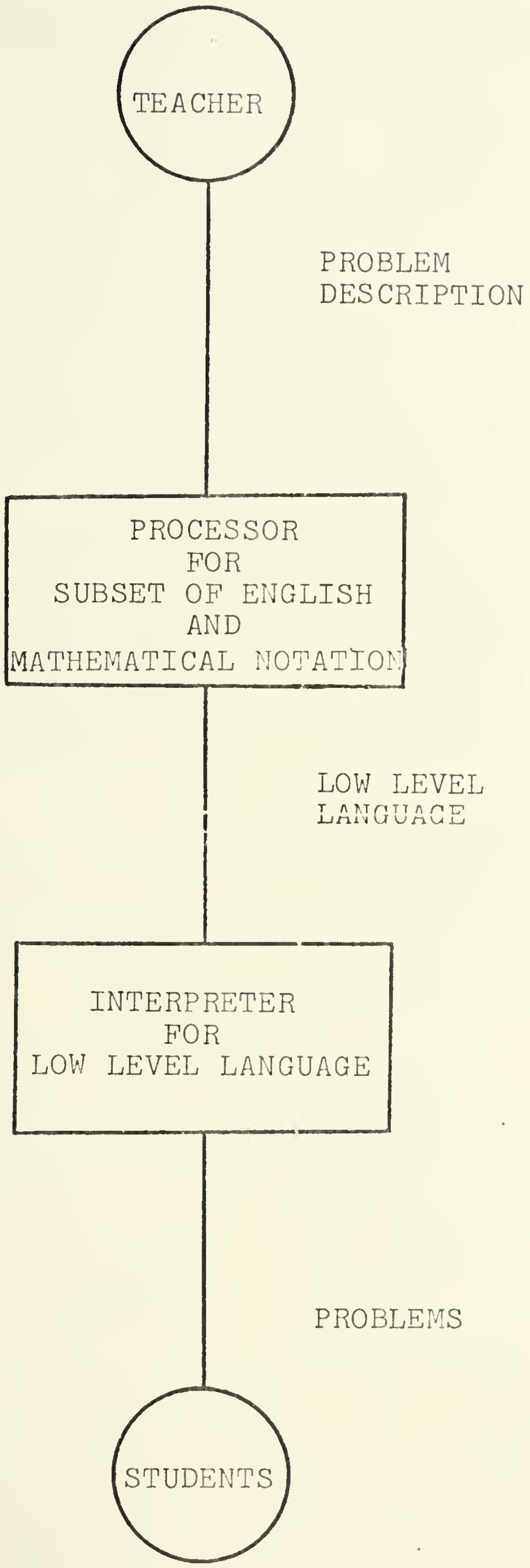
TEACHER
PROBLEM DESCRIPTION
PROCESSOR FOR SUBSET OF ENGLISH AND MATHEMATICAL NOTATION
LOW LEVEL LANGUAGE
INTERPRETER FOR LOW LEVEL LANGUAGE
PROBLEMS
STUDENTS

Figure 1. Flow of Information

The low level language was intentionally designed so that it could be easily interpreted by the machine. Although it is not normally done, a teacher who was accustomed to the system and had a knowledge about problem construction could describe his own problems if he were so inclined. The syntax for this language is described in Appendix C. An example of an addition problem with no carries in this language would look like this:

```
    (1)  X4X3X2X1 + Y3Y2Y1 = ANSWER
         X4 = A_NUMBER(0->9)
         X3 = A_NUMBER(0->9)
         X2 = A_NUMBER(0->9)
         X1 = A_NUMBER(0->9)
         Y3 = A_NUMBER(0->9-X3)
         Y2 = A_NUMBER(0->9-X2)
         Y1 = A_NUMBER(0->9-X1)
         HOR;
```

As was stated above the basis for the development of the interpreter was XPL SKELETON which is described in Ref.14. The general relations among the major procedures and GET_VALUE_OF are shown in Figure 2.

GET_VALUE_OF uses a set of array variables: SYMBOL, which holds every identifier that appears in the problem; VALUE_OF, which holds the single digit value of its corresponding identifier; NUMBER_OF, which is a character string that delimits the value its identifier can assume; FLAG, which tells if NUMBER_OF contains a valid numerical string or not; VAL_FLAG, which tells if VALUE_OF requires a random digit to be generated or not. Once the values are obtained for each identifier, HOR or VERT prints out the problem to be worked.

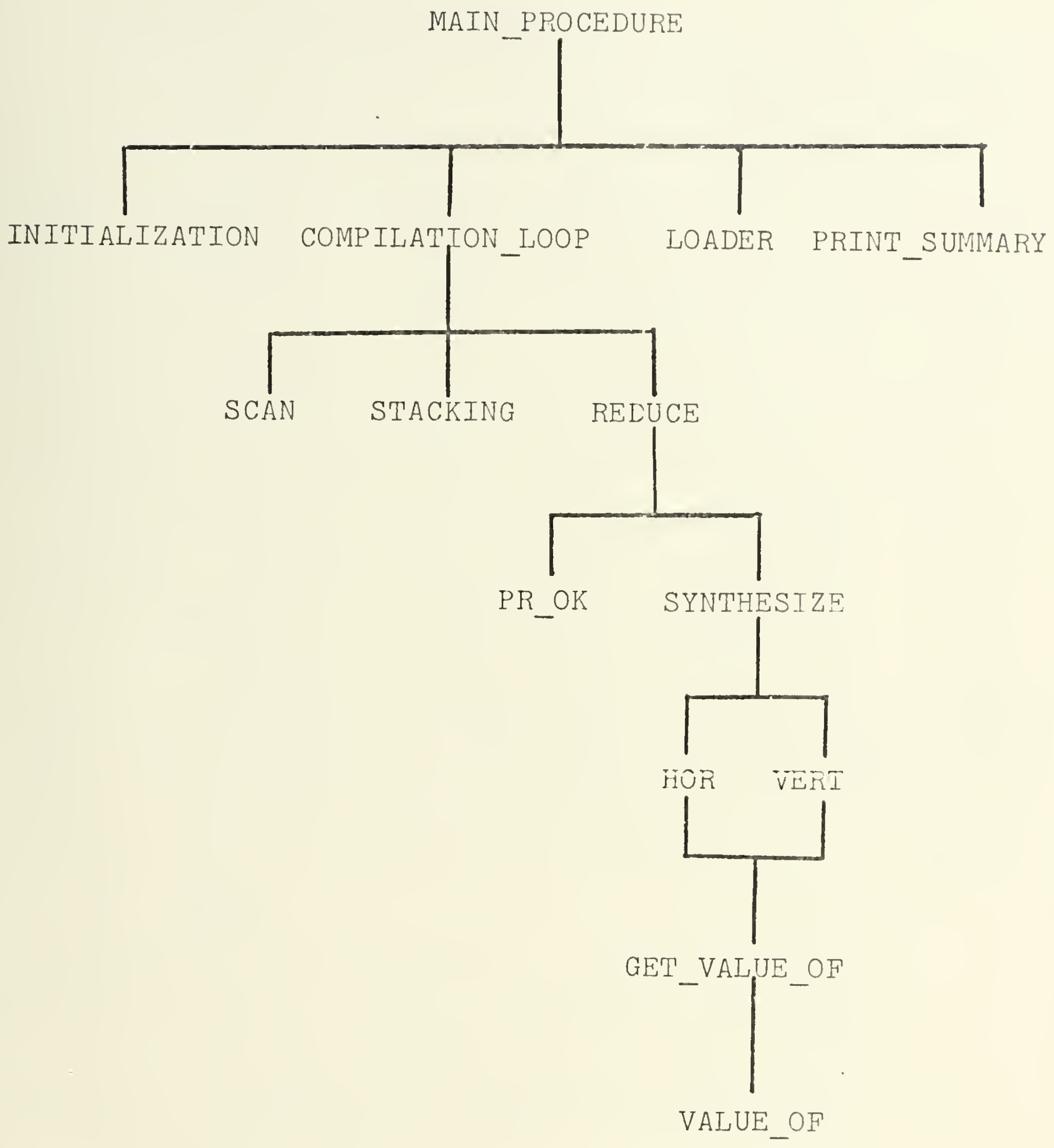

Figure 2. Relations among Major Procedures

B. PROBLEM DESCRIPTION PROCESSOR

After developing the PROBLEM DESCRIPTION INTERPRETER, a processor was developed that would accept a subset of English and mathematical notation and produce the low level language of the interpreter. The language was written for the processor with the idea that the PROBLEM DESCRIPTION PROCESSOR would be expanded to handle problem descriptions of geometry, set theory, geometric construction, and trigonometry problems if the system was ever used in conjunction with graphics terminals. Some examples of this language are:

(2) X4X3X2X1 + Y3Y2Y1 + ANSWER VERT

This is the simplest problem that can be stated. The problem would be shown as a standard addition problem of two numbers with a line drawn under them such as:

```
(3)  6037
      592
     ----
```

The following is a standard subtraction problem:

(4) X4X3X2X1 - Y3Y2Y1 = ANSWER HOR

It will be presented to the student like this:

(5) 7452 - 231 = ______

The following example

(6) W4W3W2W1 +X4X3X2X1 = ANSWER HOR CARRIES IN ONES PLACE

would generate a problem which would have a carry from ones place to tens place.

The same type of problem could be generated by:

(7) W4W3W2W1 + X4X3X2X1 = ANSWER HOR, W1 + X1 > 9

The example:

(8) Y4Y3Y2Y1 + X3X2X1 ? Y4Y3Y2Y1 + X3X2X1

requires the student to figure out what the relationship between both sides of the equation is.

(9) X4X3X2X1 + ANSWER = Z5Z4Z3Z2Z1 VERT. Z5 > 0

The student will have to perform a subtraction operation in order to find the answer. This problem specifies that the problem is to be presented vertically.

The number of constraints on the individual digits must be less than 32 and have commas separating each of them. An example which uses a number of constraints might be:

(10) X4X3X2X1 + Y4Y3Y2Y1 = ANSWER HOR. Y4 > Y3, Y3 > Y2,
Y2 >= Y1, X3≠Y3, X2 + Y2 > 9, X4 = Y4, X1 = 5.

XPL SKELETON is used as a basis for the processor in much the same way as it was for the interpreter. In SKELETON, SYNTHESIZE enters the constraint equation identifiers which have not been entered in the symbol table and then calls RESOLVE which looks at the constraints, manipulates them into a usable form, and produces code for the interpreter. The first manipulation is done by placing the constraint equations into the following order: type zero is<identifier><relation> <number>, type one is <identifier> <relation> <identifier>, type two is <identifier> + <identifier> <relation> <number>, type three which is not implemented in the problem description is <identifier> + <identifier> <relation> <identifier>. After this has been done to Example (10) above, it would look like this:

```
(11)  X4X3X2X1 + Y4Y3Y2Y1 = ANSWER HOR.
      X1 = 5
      Y4 > Y3
      Y3 > Y2
      Y2 >= Y1
      X3 ≠ Y3
      X4 = Y4
      X2 + Y2 > 9
```

The problem is again manipulated and the relations are all changed to either equal, less than, less than or equal or not equal. After this has been accomplished the problem would be:

```
(12)  X4X3X2X1 + Y4Y3Y2Y1 = ANSWER HOR
      X1 = 5
      X4 = Y4
      Y3 < Y4
      Y2 < Y3
      Y1  <= Y2
      X3 ≠ Y3
       9 < X2 + Y2
```

After this manipulation, the code for the interpreter is generated. This is done by taking the identifiers which already have been entered in the symbol table and branching to different code generating procedures depending on the type classification and its relation. After processing all of the constraints, code is generated for any digits that are not limited by constraints. This code is placed before the code that was generated by the constraint equations. The complete code file of low level language would be:

```
(13)  X4X3X2X1 + Y4Y3Y2Y1 = ANSWER
      Y4 = A_NUMBER (0->9)
      X1 = A_NUMBER (5)
      X4 = A_NUMBER (Y4)
      Y3 = A_NUMBER (0->Y4)
      Y2 = A_NUMBER (0->Y3)
      Y1 = A_NUMBER (0->Y2)
      X3 = A_NUMBER (0->Y3-1, Y3+1->9)
      X2 = A_NUMBER (9-X2->9)
      HOR;
```

If there are any errors in the problem description they will be printed out by the mechanism of SKELETON. If any conflicts between the constraint equations arise the first constraint will take precedence over the second constraint. An excerpt from a code file which has conflicts may look like this:

```
    (14)  X1 < Y1
          X2 = 5
          X4 < 6
          X1 = 9-Y1
```

In this case X1 would take on a value less than the value of Y1 instead of a value in the range from 0 to 9 minus the value of Y1.

V. CONCLUDING REMARKS

This research has demonstrated that a higher level language for describing general types of problems can be implemented. Although the system which was developed concerned itself with fourth grade arithmetic, the general approach could easily be expanded to include other areas of instruction. In general, the system represents one step toward providing a relatively easy language in which teachers can design computer assisted instruction systems which do more than present a fixed set of problems, correct them, and record the progress of the various students.

The system was designed to be included as a part of a flexible tutoring system, but it could certainly be incorporated in any computer assisted instruction system which involves general types, or classes of problems. For example, a drill and practice system could very nicely be designed around such a higher level language for problems.

The hardware is available now for large computer assisted instruction systems, but there still remains a considerable amount of software development before such systems can be economically used. This research is a contribution toward this software development.

APPENDIX A

Format description of language for communicating with the PROBLEM DESCRIPTION PROCESSOR.

The meta-symbols used in the description of the language serve the following functions:

::=	is used to indicate a definition
|	is used to indicate alternate definitions
< >	are used to enclose items which are elements of the meta-language which describe the elements of the PROBLEM DESCRIPTION PROCESSOR language

```
<teachers statement>   ::=  <teachers definition>

<teachers definition>  ::=  <problem definition> | <teachers
                            definition> <problem definition>
                            |<function definition> | <teachers
                            definition> <function definition>
                            |<definition definition> |
                            <teachers definition> <definition
                            definition>

<problem definition>   ::=  <set operations> | <geometry> |
                            <statistics and probability> |
                            <trigonometry> | <arithmetic
                            operations> <problem construction>
                            <problem constraints> |
                            <arithmetic operations> <problem
                            construction>

<arithmetic operations> ::= <expression> <relation> <expres-
                                                        sion>

<expression> ::=  <term> | <expression> + <term> | <expression>
                  - <term> | + <term> | - <term>

<term> ::= <primary> | <term> * <primary> | <term> / <primary>

<primary> ::= <number> | <identifier> | ( <expression> ) |
              ANSWER

<relation> ::= = | < | > | ≠ | ≮ | ≯ | < = | > = | ?

<problem direction> ::=  VERT | VERTICAL | HORIZ | HOR |
                         HORIZONTAL
```

```
<problem construction> ::= <problem direction> CARRIES IN
                           <identifier> PLACE | <problem
                           direction> CARRIES IN POSITION
                           <number> | <problem direction>
                           BARROWS IN <identifier> PLACE
                           <problem direction> BARROWS IN
                           POSITION <number> | <problem
                           direction>

<problem constraints>::=   <problem constraints right> |
                           <problem constraints> <problem
                           constraints right>

<problem constraints right> ::= <left problem constraint>
                                <number> | <left problem
                                constraint> <identifier>

<left problem constraint> ::= <middle constraint> <identifier>
                              <relation> | , <identifier>
                              <relation>

<middle constraint>  ::=  , <identifier> +

<set operations> ::= <union> | <intersection> | <complement>
                     | <disjoint> | <equal>

<geometry>  ::=  <metric> | <non-metric>

<statistics and probability> ::= <measure of central tendency>

<metric>  ::= <area of regions>

<non-metric> ::= <lines, points and curves> | <planes> |
                 <angles> | <polygons> | <circles> |
                 <construction>

<measure of central tendency> ::=  <average> | <range> |
                                   <mode> | <median>
```

APPENDIX 3

```
                         PROBLEM DESCRIPTION PROCESSOR

                                  R. J. WOOLS

                               COMPUTER SCIENCE
                                   STUDENT

                              NAVAL POSTGRADUATE
                                   SCHOOL

                                  CALIFORNA
                                    93940

    */

DECLARE
   MAXSYM          LITERALLY      '32',    /*  MAXSYMBOLS IN TABLE             */
   EQUALS          LITERALLY      '1',
   LESS_THAN       LITERALLY      '2',
   LESS_EQUAL      LITERALLY      '3',
   NOT_EQUAL       LITERALLY      '4';

DECLARE
   SYMBOL(MAXSYM)  CHARACTER,     /*  SYMBOL TABLE                          */
   FLAG            BIT(1),
   DIRECTION       CHARACTER,
   PROBLEM(32)     CHARACTER,
   PROB_LOC(32) BIT(16),          /* LOCATION WITHIN PROBLEM OF IDENTIFIERS */
   TYPE(32)        BIT(8),        /* TYPE OF PROBLEM CONSTRAINT             */
   DATA(64)        CHARACTER,
   AUX_DATA(64)    CHARACTER,
   DATA_PTR        FIXED,
   AUX_PTR         FIXED,
   (SY, CONPTR, NUM_ANS, NUM_QUES, POSITION) FIXED;

DECLARE NSY LITERALLY '65', NT LITERALLY '45';
DECLARE V(NSY) CHARACTER INITIAL ( '<ERROR: TOKEN = 0>', '+', '-', '*', '/',
   '(', ')', '=', '<', '>', '¬', '?', ',', 'IN', '_|_', 'HOR', 'VERT',
   'PLACE', 'HORIZ', 'ANSWER', '<MODE>', 'CARRIES', 'BARROWS', '<UNION>',
   '<EQUAL>', '<RANGE>', '<NUMBER>', 'POSITION', 'VERTICAL', '<PLANES>',
   '<ANGLES>', '<MEDIAN>', 'HORIZONAL', '<CIRCLES>', '<AVERAGE>',
   '<DISJOINT>', '<POLYGONS>', '<IDENTIFIER>', '<COMPLEMENT>',
   '<TRIGONOMETRY>', '<INTERSECTION>', '<CONSTRUCTION>', '<AREA OF REGIONS>',
```

```
   '<FUNCTION DEFINITION>', '<DEFINITION DEFINITION>',
   '<LINES, POINTS AND CURVES>', '<TERM>', '<METRIC>', '<PRIMARY>',
   '<GEOMETRY>', '<RELATION>', '<EXPRESSION>', '<NON-METRIC>',
   '<SET OPERATIONS>', '<PROBLEM DIRECTION>', '<MIDDLE CONSTRAINT>',
   '<TEACHERS STATEMENT>', '<PROBLEM DEFINITION>', '<TEACHERS DEFINITION>',
   '<PROBLEM CONSTRAINTS>', '<PROBLEM CONSTRUCTION>',
   '<ARITHMETIC OPERATIONS>', '<LEFT PROBLEM CONSTRAINT>',
   '<PROBLEM CONSTRAINTS RIGHT>', '<STATISTICS AND PROBABILITY>',
   '<MEASURE OF CENTRAL TENDENCY>');
DECLARE V_INDEX(26) BIT(8) INITIAL ( 1, 13, 14, 16, 17, 19, 21, 26, 32, 35,
   37, 37, 39, 39, 42, 42, 42, 43, 43, 43, 43, 44, 44, 45, 45, 45, 46);
DECLARE C1(NSY) BIT(92) INITIAL (
   "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 0",
   "(2) 00000 10000 00000 00001 00000 01000 00000 00300 00000 0",
   "(2) 00000 10000 00000 00001 00000 01000 00000 00100 00000 0",
   "(2) 00000 10000 00000 00001 00000 01000 00000 00100 00000 0",
   "(2) 00000 10000 00000 00001 00000 01000 00000 00100 00000 0",
   "(2) 01100 10000 00000 00001 00000 01000 00000 00100 00000 0",
   "(2) 02222 02222 22000 22020 00000 00020 00200 00000 00000 0",
   "(2) 02200 20000 00000 00002 00000 02000 00000 00200 00000 0",
   "(2) 02200 20100 00000 00002 00000 02000 00000 00200 00000 0",
   "(2) 02200 20100 00000 00002 00000 02000 00000 00200 00000 0",
   "(2) 00000 00111 00000 00000 00000 00000 00000 00000 00000 0",
   "(2) 02200 20000 00000 00002 00000 02000 00000 00200 00000 0",
   "(2) 00000 00000 00000 00000 00000 00200 00000 00100 00000 0",
   "(2) 00000 00000 00000 00000 00000 00100 00000 00100 00000 0",
   "(2) 01100 10000 00000 00001 10011 11001 11011 11111 11111 1",
   "(2) 02200 20000 00202 00002 22222 22002 22022 22222 22222 2",
   "(2) 02200 20000 00202 00002 22222 22002 22022 22222 22222 2",
   "(2) 02200 20000 00202 00002 20022 22002 22022 22222 22222 2",
   "(2) 02200 20000 00202 00002 22222 22002 22022 22222 22222 2",
   "(2) 02222 02222 22000 22020 00000 00020 00200 00000 00000 0",
   "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
   "(2) 00000 00000 00010 00000 00000 00000 00000 00000 00000 0",
   "(2) 00000 00000 00010 00000 00000 00000 00000 00000 00000 0",
   "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
   "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
   "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
   "(2) 02222 22222 22202 22022 20022 22022 22222 22222 22222 2",
   "(2) 00000 00000 00000 00000 00000 01000 00000 00000 00000 0",
   "(2) 02200 20000 00202 00002 22222 22002 22022 22222 22222 2",
   "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
   "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
   "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
   "(2) 02200 20000 00202 00002 22222 22002 22022 22222 22222 2",
   "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
   "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
   "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
```

```
    "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
    "(2) 03222 22333 33202 22122 20022 22022 22222 22222 22222 2",
    "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
    "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
    "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
    "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
    "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
    "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
    "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
    "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
    "(2) 02211 02222 22000 22020 00000 00020 00200 00000 00000 0",
    "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
    "(2) 02222 02222 22000 22020 00000 00020 00200 00000 00000 0",
    "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
    "(2) 01100 10000 00000 00001 00000 03000 00000 00300 00000 0",
    "(2) 01100 02111 11000 22020 00000 00020 00200 00000 00000 0",
    "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
    "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
    "(2) 02200 20000 00202 00002 21122 22002 22022 22222 22222 2",
    "(2) 00000 00000 00000 00000 00000 00000 00000 00100 00000 0",
    "(2) 00000 00000 00002 00000 00000 00000 00000 00000 00000 0",
    "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
    "(2) 01100 10000 00002 00001 10011 11001 11011 11111 11111 1",
    "(2) 02200 20000 00102 00002 20022 22002 22022 22222 22222 2",
    "(2) 02200 20000 00102 00002 20022 22002 22022 22222 22222 2",
    "(2) 00000 01000 00000 11010 00000 00010 00100 00000 00000 0",
    "(2) 00000 00000 00000 00000 00000 01000 00000 00100 00000 0",
    "(2) 02200 20000 00202 00002 20022 22002 22022 22222 22222 2",
    "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2",
    "(2) 02200 20000 00002 00002 20022 22002 22022 22222 22222 2");
DECLARE NC1TRIPLES LITERALLY '17';
DECLARE C1TRIPLES(NC1TRIPLES) FIXED INITIAL ( 327973, 795905, 795911, 795912,
    795913, 795914, 795915, 917797, 3277093, 3342629, 3355162, 3355173,
    3613959, 3613960, 3613961, 3613962, 3613963, 3801381);
DECLARE PRTB(71) FIXED INITIAL (0, 3109, 1341, 10, 8, 9, 0, 10, 0, 10, 0, 0,
    0, 0, 907349285, 907414821, 0, 0, 0, 0, 0, 0, 907349275, 907414811, 62, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 62, 0, 0, 0, 0, 0, 0, 58, 0, 58, 0, 0, 13057,
    13058, 1, 2, 0, 0, 11779, 11780, 0, 0, 14117, 3109, 13106, 0, 0, 0, 58, 0,
    0, 15676, 61, 59, 0, 0, 0);
DECLARE PRDTB(71) BIT(8) INITIAL (0, 52, 25, 30, 33, 34, 27, 31, 28, 32, 29,
    35, 44, 41, 36, 38, 43, 26, 70, 53, 57, 69, 37, 39, 48, 23, 42, 63, 64,
    71, 45, 66, 68, 56, 65, 49, 24, 55, 13, 54, 67, 61, 5, 4, 7, 6, 62, 16,
    17, 18, 19, 15, 58, 21, 22, 20, 11, 50, 51, 14, 59, 8, 40, 3, 2, 1, 9, 10,
    47, 46, 12, 60);
DECLARE HDTB(71) BIT(8) INITIAL (0, 55, 48, 50, 50, 50, 50, 50, 50, 50, 50,
    50, 54, 54, 60, 60, 54, 48, 65, 53, 53, 65, 60, 60, 63, 48, 54, 52, 52,
    65, 54, 52, 65, 53, 52, 63, 48, 53, 57, 53, 52, 47, 58, 58, 58, 58, 52,
    51, 51, 51, 51, 51, 49, 46, 46, 46, 57, 62, 62, 61, 49, 57, 60, 58, 58,
```

```
    56, 57, 57, 59, 59, 57, 64);
DECLARE PRLENGTH(71) BIT(8) INITIAL (0, 3, 3, 2, 2, 2, 1, 2, 1, 2, 1, 1, 1,
    1, 5, 5, 1, 1, 1, 1, 1, 1, 5, 5, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1,
    1, 1, 1, 1, 2, 1, 2, 1, 1, 3, 3, 2, 2, 1, 1, 3, 3, 1, 1, 3, 3, 3, 1, 1, 1,
    2, 1, 1, 3, 2, 2, 1, 1, 1);
DECLARE CONTEXT_CASE(71) BIT(8) INITIAL (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
DECLARE LEFT_CONTEXT(0) BIT(8) INITIAL ( 0);
DECLARE LEFT_INDEX(20) BIT(8) INITIAL ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 1):
DECLARE CONTEXT_TRIPLE(0) FIXED INITIAL ( 0);
DECLARE TRIPLE_INDEX(20) BIT(8) INITIAL ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 1);
DECLARE PR_INDEX(65) BIT(8) INITIAL ( 1, 2, 2, 2, 2, 2, 3, 7, 9, 11, 11, 12,
    12, 12, 12, 13, 14, 16, 17, 18, 19, 19, 19, 20, 21, 22, 26, 26, 27, 28,
    29, 30, 31, 32, 33, 34, 35, 37, 38, 39, 40, 41, 42, 44, 46, 47, 52, 53,
    56, 57, 59, 60, 61, 62, 63, 63, 63, 65, 66, 67, 68, 68, 68, 70, 71, 72):

/*  END OF CARDS PUNCHED BY SYNTAX                                        */

DECLARE (I1, I2, I3, I4) FIXED;
/*  DECLARATIONS FOR THE SCANNER                                          */

/* TOKEN IS THE INDEX INTO THE VOCABULARY V() OF THE LAST SYMBOL SCANNED,
   CP IS THE POINTER TO THE LAST CHARACTER SCANNED IN THE CARDIMAGE,
   BCD IS THE LAST SYMBOL SCANNED (LITERAL CHARACTER STRING). */
DECLARE (TOKEN, CP) FIXED, BCD CHARACTER;

/* SET UP SOME CONVENIENT ABBREVIATIONS FOR PRINTER CONTROL */
DECLARE EJECT_PAGE LITERALLY 'OUTPUT(1) = PAGE',
   PAGE CHARACTER INITIAL ('1'), DOUBLE CHARACTER INITIAL ('0'),
   DOUBLE_SPACE LITERALLY 'OUTPUT(1) = DOUBLE',
   X70 CHARACTER INITIAL ('
                 ');

/* LENGTH OF LONGEST SYMBOL IN V */
DECLARE (RESERVED_LIMIT, MARGIN_CHOP) FIXED;

/* CHARTYPE() IS USED TO DISTINGUISH CLASSES OF SYMBOLS IN THE SCANNER.
   TX() IS A TABLE USED FOR TRANSLATING FROM ONE CHARACTER SET TO ANOTHER.
   CONTROL() HOLDS THE VALUE OF THE COMPILER CONTROL TOGGLES SET IN $ CARDS.
   NOT_LETTER_OR_DIGIT() IS SIMILIAR TO CHARTYPE() BUT USED IN SCANNING
   IDENTIFIERS ONLY.

   ALL ARE USED BY THE SCANNER AND CONTROL() IS SET THERE.
*/
```

```
DECLARE (CHARTYPE, TX) (255) BIT(8),
        (CONTROL, NOT_LETTER_OR_DIGIT)(255) BIT(1);

/* ALPHABET CONSISTS OF THE SYMBOLS CONSIDERED ALPHABETIC IN BUILDING
   IDENTIFIERS     */
DECLARE ALPHABET CHARACTER INITIAL ('ABCDEFGHIJKLMNOPQRSTUVWXYZ_$@#');

/* BUFFER HOLDS THE LATEST CARDIMAGE,
   TEXT HOLDS THE PRESENT STATE OF THE INPUT TEXT
   (NOT INCLUDING THE PORTIONS DELETED BY THE SCANNER),
   TEXT_LIMIT IS A CONVENIENT PLACE TO STORE THE POINTER TO THE END OF TEXT,
   CARD_COUNT IS INCREMENTED BY ONE FOR EVERY SOURCE CARD READ,
   ERROR_COUNT TABULATES THE ERRORS AS THEY ARE DETECTED,
   SEVERE_ERRORS TABULATES THOSE ERRORS OF FATAL SIGNIFICANCE.
*/
DECLARE (BUFFER, TEXT) CHARACTER,
   (TEXT_LIMIT, CARD_COUNT, ERROR_COUNT, SEVERE_ERRORS, PREVIOUS_ERROR) FIXED
   ;

/* NUMBER_VALUE CONTAINS THE NUMERIC VALUE OF THE LAST CONSTANT SCANNED,
*/
DECLARE NUMBER_VALUE FIXED;

/* EACH OF THE FOLLOWING CONTAINS THE INDEX INTO V() OF THE CORRESPONDING
   SYMBOL.   WE ASK:     IF TOKEN = IDENT     ETC.     */
DECLARE (IDENT, NUMBER, DIVIDE, EOFILE, STRING, LETTER)  FIXED ;

/* STOPIT() IS A TABLE OF SYMBOLS WHICH ARE ALLOWED TO TERMINATE THE ERROR
   FLUSH PROCESS.  IN GENERAL THEY ARE SYMBOLS OF SUFFICIENT SYNTACTIC
   HIERARCHY THAT WE EXPECT TO AVOID ATTEMPTING TO START CHECKING AGAIN
   RIGHT INTO ANOTHER ERROR PRODUCING SITUATION.  THE TOKEN STACK IS ALSO
   FLUSHED DOWN TO SOMETHING ACCEPTABLE TO A STOPIT() SYMBOL.
   FAILSOFT IS A BIT WHICH ALLOWS THE COMPILER ONE ATTEMPT AT A GENTLE
   RECOVERY.   THEN IT TAKES A STRONG HAND.   WHEN THERE IS REAL TROUBLE
   COMPILING IS SET TO FALSE, THEREBY TERMINATING THE COMPILATION.
*/
DECLARE STOPIT(100) BIT(1)., (FAILSOFT, COMPILING) BIT(1);

DECLARE S CHARACTER;  /* A TEMPORARY USED VARIOUS PLACES */

/* THE ENTRIES IN PRMASK() ARE USED TO SELECT OUT PORTIONS OF CODED
   PRODUCTIONS AND THE STACK TOP FOR COMPARISON IN THE ANALYSIS ALGORITHM */
DECLARE PRMASK(5) FIXED INITIAL (0, 0, "FF", "FFFF", "FFFFFF", "FFFFFFFF");


/*THE PROPER SUBSTRING OF POINTER IS USED TO PLACE AN  |  UNDER THE POINT
   OF DETECTION OF AN ERROR DURING CHECKING.  IT MARKS THE LAST CHARACTER
   SCANNED.  */
```

```
    DECLARE POINTER CHARACTER INITIAL ('
                                                  |');
    DECLARE CALLCOUNT(20) FIXED   /* COUNT THE CALLS OF IMPORTANT PROCEDURES */
       INITIAL(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);

    /* RECORD THE TIMES OF IMPORTANT POINTS DURING CHECKING */
    DECLARE CLOCK(5) FIXED;


    /* COMMONLY USED STRINGS */
    DECLARE X1 CHARACTER INITIAL(' '), X4 CHARACTER INITIAL('    ');
    DECLARE PERIOD CHARACTER INITIAL ('.');

    /* TEMPORARIES USED THROUGHOUT THE COMPILER */
    DECLARE (I, J, K, L) FIXED;

    DECLARE TRUE LITERALLY '1', FALSE LITERALLY '0', FOREVER LITERALLY 'WHILE 1';

    /*  THE STACKS DECLARED BELOW ARE USED TO DRIVE THE SYNTACTIC
       ANALYSIS ALGORITHM AND STORE INFORMATION RELEVANT TO THE INTERPRETATION
       OF THE TEXT.  THE STACKS ARE ALL POINTED TO BY THE STACK POINTER SP.  */

    DECLARE STACKSIZE LITERALLY '75';  /* SIZE OF STACK  */
    DECLARE PARSE_STACK (STACKSIZE) BIT(8); /* TOKENS OF THE PARTIALLY PARSED
                                               TEXT */
    DECLARE VAR (STACKSIZE) CHARACTER;/* EBCDIC NAME OF ITEM */
    DECLARE FIXV (STACKSIZE) FIXED;   /* FIXED (NUMERIC) VALUE */

    /* SP POINTS TO THE RIGHT END OF THE REDUCIBLE STRING IN THE PARSE STACK,
       MP POINTS TO THE LEFT END, AND
       MPP1 = MP+1.
    */
    DECLARE (SP, MP, MPP1) FIXED;


    /*               P R O C E D U R E S                               */


 PAD:
    PROCEDURE (STRING, WIDTH) CHARACTER;
       DECLARE STRING CHARACTER, (WIDTH, L) FIXED;

       L = LENGTH(STRING);
       IF L >= WIDTH THEN RETURN STRING;
       ELSE RETURN STRING || SUBSTR(X70, 0, WIDTH-L);
    END PAD;

 I_FORMAT:
```

```
   PROCEDURE (NUMBER, WIDTH) CHARACTER;
      DECLARE (NUMBER, WIDTH, L) FIXED, STRING CHARACTER;

      STRING = NUMBER;
      L = LENGTH(STRING);
      IF L >= WIDTH THEN PETURN STRING;
      ELSE RETURN SUBSTR(X70, 0, WIDTH-L) || STRING;
   END I_FORMAT;

ERROR:
   PROCEDURE(MSG, SEVERITY);
      /* PRINTS AND ACCOUNTS FOR ALL ERROR MESSAGES */
      /* IF SEVERITY IS NOT SUPPLIED, C IS ASSUMED */
      DECLARE MSG CHARACTER, SEVERITY FIXED;
      ERROR_COUNT = ERROR_COUNT + 1;
      /* IF LISTING IS SUPPRESSED, FORCE PRINTING OF THIS LINE */
      IF ¬ CONTROL(BYTE('L')) THEN
         OUTPUT = I_FORMAT (CARD_COUNT, 4) || ' |' || BUFFER || '|';
      OUTPUT = SUBSTR(POINTER, TEXT_LIMIT-CP+MARGIN_CHOP);
      OUTPUT = '*** ERROR, ' || MSG || ' . ***';
      IF ERROR_COUNT > 1 THEN OUTPUT = '***  LAST PREVIOUS ERROR WAS DETECTED '
         || 'ON LINE ' || PREVIOUS_ERROR || ' .  ***';
      DOUBLE_SPACE;
      PREVIOUS_ERROR = CARD_COUNT;
      IF SEVERITY > 0 THEN
         IF SEVERE_ERRORS > 25 THEN
            DO;
               OUTPUT = '*** TOO MANY SEVERE ERRORS, CHECKING ABORTED ***';
               COMPILING = FALSE;
            END;
         ELSE SEVERE_ERRORS = SEVERE_ERRORS + 1;
   END ERROR;


  /*                    CARD IMAGE HANDLING PROCEDURE                  */

GET_CARD:
   PROCEDURE;
      /* DOES ALL CARD READING AND LISTING                              */
      DECLARE I FIXED, (TEMP, TEMPO, REST) CHARACTER, READING BIT(1);
            BUFFER = INPUT;
            IF LENGTH(BUFFER) = 0 THEN
               DO; /* SIGNAL FOR EOF */
                  CALL ERROR('EOF  MISSING OR COMMENT STARTING IN COLUMN 1.',1);
                  BUFFER = PAD (' /*''/* */ EOF;END;EOF', 80);
               END;
            ELSE CARD_COUNT = CARD_COUNT + 1:  /* USED TO PRINT ON LISTING */
      IF MARGIN_CHOP ≥ 0 THEN
```

```
          DO; /* THE MARGIN CONTROL FROM DOLLAR | */
             I = LENGTH(BUFFER) - MARGIN_CHOP;
             REST = SUBSTR(BUFFER, I);
             BUFFER = SUBSTR(BUFFER, 0, I);
          END;
       ELSE REST = '';
       TEXT = BUFFER;
       TEXT_LIMIT = LENGTH(TEXT) - 1;
       IF CONTROL(BYTE('M')) THEN OUTPUT = BUFFER;
       ELSE IF CONTROL(BYTE('L')) THEN
          OUTPUT = I_FORMAT (CARD_COUNT, 4) || ' |' || BUFFER || '|' || REST;
       CP = 0;
    END GET_CARD;


    /*                   THE SCANNER PROCEDURES                  */


CHAR:
    PROCEDURE:
       /* USED FOR STRINGS TO AVOID CARD BOUNDARY PROBLEMS */
       CP = CP + 1;
       IF CP <= TEXT_LIMIT THEN RETURN;
       CALL GET_CARD;
    END CHAR;


SCAN:
    PROCEDURE:
       DECLARE (S1, S2) FIXED;
       DECLARE LSTRNGM CHARACTER INITIAL(' STRING TOO LONG '), STRDELIM CHARACTER
          ,LOOP BIT(8);
       CALLCOUNT(3) = CALLCOUNT(3) + 1:
       FAILSOFT = TRUE;
       BCD = '';  NUMBER_VALUE = 0;
    SCAN1:
       DO FOREVER;
          IF CP > TEXT_LIMIT THEN CALL GET_CARD:
          ELSE
             DO; /* DISCARD LAST SCANNED VALUE */
                TEXT_LIMIT = TEXT_LIMIT - CP;
                TEXT = SUBSTR(TEXT, CP);
                CP = 0;
             END;
          /*  BRANCH ON NEXT CHARACTER IN TEXT                     */
          DO CASE CHARTYPE(BYTE(TEXT));

             /*  CASE 0   */
```

```
/* ILLEGAL CHARACTERS FALL HERE  */
CALL ERROR ('ILLEGAL CHARACTER: ' || SUBSTR(TEXT, 0, 1));

/*  CASE 1  */

/*  BLANK  */
DO;
   CP = 1;
   DO WHILE BYTE(TEXT, CP) = BYTE(' ') & CP <= TEXT_LIMIT;
      CP = CP + 1;
   END;
   CP = CP - 1;
END;

/*  CASE 2  */

DO FOREVER;  /* A LETTER:  IDENTIFIERS AND RESERVED WORDS */
TOKEN = LETTER;
   DO CP = CP + 1 TO TEXT_LIMIT;
      IF NOT_LETTER_OR_DIGIT(BYTE(TEXT, CP)) THEN
         DO;  /* END OF IDENTIFIER  */
            IF CP > 0 THEN BCD = BCD || SUBSTR(TEXT, 0, CP);
            S1 = LENGTH(BCD);
            IF S1 > 1 THEN IF S1 <= RESERVED_LIMIT THEN
               /* CHECK FOR RESERVED WORDS */
               DO I = V_INDEX(S1-1) TO V_INDEX(S1) - 1;
                  IF BCD = V(I) THEN
                     DO;
                        TOKEN = I;
                        RETURN;
                     END;
               END;
            /*  RESERVED WORDS EXIT HIGHER: THEREFORE <IDENTIFIER>*/
            TOKEN = IDENT;
            RETURN;
         END;
   END;
   /*  END OF CARD  */
   BCD = BCD || TEXT;
   CALL GET_CARD;
   CP = -1;
END;

/*  CASE 3  */

DO;       /*  DIGIT:  A NUMBER  */
   TOKEN = NUMBER;
```

```
      DO FOREVER;
         DO CP = CP TO TEXT_LIMIT;
            S1 = BYTE(TEXT, CP);
            IF S1 < "F0" THEN DO;
               PROBLEM(CONPTR) = PROBLEM(CONPTR) || NUMBER_VALUE;
               RETURN;
            END;
            NUMBER_VALUE = 10*NUMBER_VALUE + S1 - "F0";
         END;
         CALL GET_CARD;
      END;
   END;

   /*  CASE 4  */

   DO;      /*  A /:  MAY BE DIVIDE OR START OF COMMENT  */
      CALL CHAR;
      IF BYTE(TEXT, CP) ¬= BYTE('*') THEN
         DO;
            TOKEN = DIVIDE;
            PROBLEM(CONPTR) = PROBLEM(CONPTR) || '/';
            RETURN;
         END;
      /* WE HAVE A COMMENT  */
      S1, S2 = BYTE(' ');
      DO WHILE S1 ¬= BYTE('*') | S2 ¬= BYTE('/');
         IF S1 = BYTE('$') THEN
            DO;  /* A CONTROL CHARACTER  */
               CONTROL(S2) = ¬ CONTROL(S2);
               IF S2 = BYTE('T') THEN CALL TRACE;
               ELSE IF S2 = BYTE('U') THEN CALL UNTRACE;
               ELSE IF S2 = BYTE('|') THEN
                  IF CONTROL(S2) THEN
                     MARGIN_CHOP = TEXT_LIMIT - CP + 1;
                  ELSE
                     MARGIN_CHOP = 0;
            END;
         S1 = S2;
         CALL CHAR;
         S2 = BYTE(TEXT, CP);
      END;
   END;

   /*  CASE 5  */

   DO;      /*  SPECIAL CHARACTERS  */
      TOKEN = TX(BYTE(TEXT));
      IF SUBSTR(TEXT,0,1) = ',' THEN DO;
```

```
                    FLAG = TRUE;
                    CONPTR = CONPTR + 1;
                    IF CONPTR > 32 THEN CALL ERROR(' CONSTRAINTS EXCEED 31 ',
                       1);
                    END;
                    ELSE PROBLEM(CONPTR) = PROBLEM(CONPTR) || SUBSTR(TEXT,0,1);
                CP = 1;
                RETURN;
              END;

          END;     /* OF CASE ON CHARTYPE  */
          CP = CP + 1;  /* ADVANCE SCANNER AND RESUME SEARCH FOR TOKEN  */
        END;
      END SCAN;


   /*                          TIME AND DATE                                    */

   PRINT_TIME:
      PROCEDURE (MESSAGE, T);
         DECLARE MESSAGE CHARACTER, T FIXED;
         MESSAGE = MESSAGE || T/360000 || ':' || T MOD 360000 / 6000 || ':'
            || T MOD 6000 / 100 || '.';
         T = T MOD 100;  /* DECIMAL FRACTION  */
         IF T < 10 THEN MESSAGE = MESSAGE || '0';
         OUTPUT = MESSAGE || T || '.';
      END PRINT_TIME;

   PRINT_DATE_AND_TIME:
      PROCEDURE (MESSAGE, D, T);
         DECLARE MESSAGE CHARACTER, (D, T, YEAR, DAY, M) FIXED;
         DECLARE MONTH(11) CHARACTER INITIAL ('JANUARY', 'FEBRUARY', 'MARCH',
            'APRIL', 'MAY', 'JUNE', 'JULY', 'AUGUST', 'SEPTEMBER', 'OCTOBER',
            'NOVEMBER', 'DECEMBER'),
         DAYS(12) FIXED INITIAL (0, 31, 60, 91, 121, 152, 182, 213, 244, 274,
            305, 335, 366);
         YEAR = D/1000 + 1900;
         DAY = D MOD 1000;
         IF (YEAR & "3") ¬= 0 THEN IF DAY > 59 THEN DAY = DAY + 1; /* ¬ LEAP YEAR*/
         M = 1;
         DO WHILE DAY > DAYS(M);  M = M + 1;  END;
         CALL PRINT_TIME(MESSAGE || MONTH(M-1) || X1 || DAY-DAYS(M-1) ||  ', '
            || YEAR || '.  CLOCK TIME = ', T);
      END PRINT_DATE_AND_TIME;


   /*                          INITIALIZATION                                   */
```

```
INITIALIZATION:
   PROCEDURE;
      EJECT_PAGE;
   CALL PRINT_DATE_AND_TIME ('       SYNTAX CHECK -- NAVAL POSTGRADUATE SCHOOL --
 TEACHER''''S_ROLE  VERSION OF ',DATE_OF_GENERATION,TIME_OF_GENERATION);
      DOUBLE_SPACE;
      CALL PRINT_DATE_AND_TIME ('TODAY IS ', DATE, TIME);
      DOUBLE_SPACE;
      DO I = 1 TO NT;
         S = V(I);
         IF S = '<NUMBER>' THEN NUMBER = I;  ELSE
         IF S = '<IDENTIFIER>' THEN IDENT = I;  ELSE
         IF S = '<STRING>' THEN STRING = I; ELSE
         IF S = '<LETTER>' THEN LETTER = I; ELSE
         IF S = '/' THEN DIVIDE = I; ELSE
         IF S = '_|_' THEN EOFILE = I;  ELSE
         ;
      END;
      IF IDENT = NT THEN RESERVED_LIMIT = LENGTH(V(NT-1));
      ELSE RESERVED_LIMIT = LENGTH(V(NT));
      V(EOFILE) = 'EOF';
      STOPIT(EOFILE) = TRUE;
      CHARTYPE(BYTE(' ')) = 1;
      DO I = 0 TO 255;
         NOT_LETTER_OR_DIGIT(I) = TRUE;
      END;
      DO I = 0 TO LENGTH(ALPHABET) - 1;
         J = BYTE(ALPHABET, I);
         TX(J) = I;
         NOT_LETTER_OR_DIGIT(J) = FALSE;
         CHARTYPE(J) = 2;
      END;
      DO I = 0 TO 9;
         J = BYTE('0123456789', I);
         NOT_LETTER_OR_DIGIT(J) = FALSE;
         CHARTYPE(J) = 3;
      END;
      DO I = V_INDEX(0) TO V_INDEX(1) - 1;
         J = BYTE(V(I));
         TX(J) = I:
         CHARTYPE(J) = 5;
      END:
      CHARTYPE(BYTE('/')) = 4;
      /* FIRST SET UP GLOBAL VARIABLES CONTROLLING SCAN, THEN CALL IT */
      CP = 0:  TEXT_LIMIT = -1;
      TEXT = '';
      CONTROL(BYTE('L')) = TRUE:
      CONPTR, SY, NUM_ANS, NUM_QUES = 0;
```

```
         CALL SCAN;
         /* INITIALIZE THE PARSE STACK */
         SP = 1;  PARSE_STACK(SP) = EOFILE;
      END INITIALIZATION;

   LOOKUP: PROCEDURE(C);
      DECLARE C CHARACTER, I FIXED;
      DO I = 1 TO SY;
         IF SYMBOL(I) = C THEN RETURN I;
      END;
      RETURN 0;
      END LOOKUP;

   ENTER: PROCEDURE(C);
      DECLARE C CHARACTER;
      SY = SY + 1;
      IF SY > MAXSYM THEN DO;
         CALL ERROR(' SYMBOL TABLE OVERFLOW ',2);
         CALL EXIT;
      END;
      SYMBOL(SY) = C;
      RETURN SY;
      END ENTER;

   TRACER: PROCEDURE(C) FIXED;
      DECLARE C FIXED, L CHARACTER;
      L = ' ';
      DO I1 = MP TO SP;
         L = L||V(PARSE_STACK(I1))||' ';
      END;
      OUTPUT = '('||C||' )'||L;
      END TRACER;

   FIND: PROCEDURE(C, L);
      DECLARE (C,S) CHARACTER, (L,I,J,K) FIXED;
      DO I3 = 1 TO SY;
         S = SYMBOL(I3);
         J = LENGTH(S);
         DO I4 = 0 TO J-L;
            IF SUBSTR(S,I4,L) = C THEN RETURN;
         END;
      END;
      I3 = 0;
      RETURN;
      END FIND;

   CODE1: PROCEDURE(C1, C2);
      DECLARE (C1, C2) CHARACTER;
```

```
    DECLARE (OUT1, OUT2) CHARACTER;
    OUT1 = C1 || '=A_NUMBER(';
    OUT2 = C2 || ')';
    AUX_PTR = AUX_PTR + 1;
    AUX_DATA(AUX_PTR) = OUT1 || OUT2;
    END CODE1;

CODE2: PROCEDURE(C1, C2);
    DECLARE (C1, C2) CHARACTER;
    DECLARE (OUT1, OUT2) CHARACTER;
    OUT1 = C1 || '=A_NUMBER(0->';
    OUT2 = C2 || ')';
    AUX_PTR = AUX_PTR + 1;
    AUX_DATA(AUX_PTR) = OUT1 || OUT2;
    END CODE2;

CODE3: PROCEDURE(C1, C2);
    DECLARE (C1, C2) CHARACTER;
    DECLARE (OUT1, OUT2) CHARACTER;
    OUT1 = C1 || '=A_NUMBER(';
    OUT2 = C2 || '->9)';
    AUX_PTR = AUX_PTR + 1;
    AUX_DATA(AUX_PTR) = OUT1 || OUT2;
    END CODE3;

CODE4: PROCEDURE(C1, C2, C3);
    DECLARE (C1, C2, C3) CHARACTER;
    DECLARE (OUT1, OUT2) CHARACTER;
    OUT1 = C1 || '=A_NUMBER(0->';
    OUT2 = C2 || ',' || C3 || '->9)';
    AUX_PTR = AUX_PTR + 1;
    AUX_DATA(AUX_PTR) = OUT1 || OUT2;
    END CODE4;

RESOLVE: PROCEDURE;
    DECLARE (TEMPROB , PROB_TEMP, OUTPUT1) CHARACTER, TEMPLOC BIT(16),
       TEMPTYPE BIT(8), TYPE_CNT(4) FIXED;
    DO I = 0 TO 3;
       TYPE_CNT(I) = 0;
    END;
    IF CONPTR = 1 THEN GO TO INSERT_CODE;
    IF CONPTR > 2 THEN DO J = 2 TO CONPTR - 1;
       DO I = J+1 TO CONPTR;
          IF TYPE(J) > TYPE(I)   THEN DO;
             TEMPTYPE = TYPE(J);
             TEMPROB = PROBLEM(J);
             TYPE(J) = TYPE(I);
             PROBLEM(J) = PROBLEM(I);
```

```
                TYPE(I)    = TEMPTYPE;
                PROBLEM(I)    = TEMPROB;
             END:
          END;
       END;
       DO I = 2 TO CONPTR;
          L = LENGTH(PROBLEM(I));
          DO J = 0 TO L - 1;
             IF BYTE(PROBLEM(I),J) = BYTE('?') THEN DO:
                CALL ERROR(' QUESTION MARK APEARS IN A CONSTRAINT EQUATION',1);
                NUM_ANS, NUM_QUES = 0;
                RETURN;
             END:
             IF BYTE(PROBLEM(I),J) = BYTE('=') THEN DO;
                PROB_LOC(I) = SHL(J,8) | EQUALS;
                GO TO QUIT;
             END:
             ELSE IF BYTE(PROBLEM(I),J) = BYTE('<') THEN DO:
                IF BYTE(PROBLEM(I),J+1) = BYTE('=') THEN DO;
                   PROB_LOC(I) = SHL(J,8) | LESS_EQUAL;
                   GO TO QUIT:
                END;
                ELSE DO:
                   PROB_LOC(I) = SHL(J,8) | LESS_THAN;
                   GO TO QUIT;
                END;
             END;
             ELSE IF BYTE(PROBLEM(I),J) = BYTE('>') THEN DO;
                IF BYTE(PROBLEM(I),J+1) = BYTE('=') THEN DO;
                   PROB_LOC(I) = SHL(L-J-2,8) | LESS_EQUAL;
                   TEMPROB = SUBSTR(PROBLEM(I),0,2);
                   PROB_TEMP = SUBSTR(PROBLEM(I),J+2);
                   PROBLEM(I) = PROB_TEMP || '<=' || TEMPROB;
                   GO TO QUIT:
                END;
                ELSE DO;
                   PROB_LOC(I) = SHL(L-J-1,8) | LESS_THAN;
                   TEMPROB = SUBSTR(PROBLEM(I),0,J);
                   PROB_TEMP  = SUBSTR(PROBLEM(I),J+1);
                   PROBLEM(I) = PROB_TEMP || '<' || TEMPROB;
                   GO TO QUIT;
                END;
             END;
             ELSE IF J <= L - 2 THEN DO;
                IF SUBSTR(PROBLEM(I),J,2) = '¬=' THEN DO;
                   PROB_LOC(I) = SHL(J,8) | NOT_EQUAL;
                   GO TO QUIT;
                END;
```

```
          ELSE IF SUBSTR(PROBLEM(I),J,2) = '¬<' THEN DO:
             PROB_LOC(I) = SHL(L-J-2,8) | LESS_EQUAL;
             TEMPROB =  SUBSTR(PROBLEM(I),0,J);
             PROB_TEMP  =  SUBSTR(PROBLEM(I),J+2);
             PROBLEM(I) = PROB_TEMP || '<=' || TEMPROB;
             GO TO QUIT;
          END;
          ELSE IF SUBSTR(PROBLEM(I),J,2) = '¬>' THEN DO;
             PROB_LOC(I) = SHL(J,8) | LESS_EQUAL;
             TEMPROB =  SUBSTR(PROBLEM(I),0,J);
             PROB_TEMP = SUBSTR(PROBLEM(I),J+2);
             PROBLEM(I) = TEMPROB||'<='||PROB_TEMP;
             GO TO QUIT;
          END;
       END;
    END;
    QUIT: IF TYPE(I) = 0 THEN TYPE_CNT(0) = TYPE_CNT(0) + 1; ELSE
    IF TYPE(I) = 1 THEN TYPE_CNT(1) = TYPE_CNT(1) + 1; ELSE
    IF TYPE(I) = 2 THEN TYPE_CNT(2) = TYPE_CNT(2) + 1; ELSE
    IF TYPE(I) = 3 THEN TYPE_CNT(3) = TYPE_CNT(3) + 1;
 END;
 I1 = 2;
 DO K = 0 TO 3;
    I2 = TYPE_CNT(K) + I1 -1;
    IF I2 - I1 > 1 THEN DO J = I1 TO I2 - 1;
       DO I = J+1 TO I2;
          IF (PROB_LOC(J) & "FF") > (PROB_LOC(I)   & "FF") THEN DO;
             TEMPLOC = PROB_LOC(J);
             TEMPROB = PROBLEM(J);
             TEMPTYPE = TYPE(J);
             PROB_LOC(J) = PROB_LOC(I);
             PROBLEM(J) = PROBLEM(I);
             TYPE(J) = TYPE(I);
             PROB_LOC(I)    = TEMPLOC;
             PROBLEM(I)    = TEMPROB;
             TYPE(I) = TEMPTYPE;
          END;
       END;
    END;
    I1 = I2 + 1;
 END;
 INSERT_CODE:DATA_PTR = DATA_PTR + 1;
 DATA(DATA_PTR) = PROBLEM(1);
 DO I = 2 TO CONPTR;

    IF TYPE(I) = 0 THEN DO;

       IF (PROB_LOC(I) & "FF") = 1 THEN DO:
```

```
    TEMPROB = SUBSTR(PROBLEM(I),0,2);
    I1 = LOOKUP(TEMPROB);
    IF I1 = 0 THEN CALL ENTER(TEMPROB);
    CALL CODE1(TEMPROB, SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+1));
END;

ELSE IF (PROB_LOC(I) & "FF") = 2 THEN DO;
    TEMPROB = SUBSTR(PROBLEM(I),0,2);
    IF BYTE(TEMPROB,0) > "F0" THEN DO;
        TEMPROB = SUBSTR(PROBLEM(I),0,SHR(PROB_LOC(I),8));
        PROB_TEMP = SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+1);
        CALL CODE3(PROB_TEMP,TEMPROB);
        CALL ENTER(PROB_TEMP);
    END;
    ELSE DO;
        I1 = LOOKUP(TEMPROB);
        IF I1 = 0 THEN CALL ENTER(TEMPROB);
        I1 = 0;
        DO J = SHR(PROB_LOC(I),8)+1 TO LENGTH(PROBLEM(I))-1;
            I1 = I1*10 + BYTE(SUBSTR(PROBLEM(I),J,1)) - "F0";
        END;
        PROB_TEMP = I1 - 1;
        CALL CODE2(TEMPROB, PROB_TEMP);
    END;
END;

ELSE IF (PROB_LOC(I) & "FF") = 3 THEN DO;
    TEMPROB = SUBSTR(PROBLEM(I),0,2);
    IF BYTE(TEMPROB,0) > "F0" THEN DO;
        TEMPROB = SUBSTR(PROBLEM(I),0,SHR(PROB_LOC(I),8));
        PROB_TEMP = SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+2);
        CALL CODE3(PROB_TEMP,TEMPROB);
        CALL ENTER(PROB_TEMP);
    END;
    ELSE DO;
        I1 = LOOKUP(TEMPROB);
        IF I1 = 0 THEN DO;
            CALL CODE2(TEMPROB,SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+2));
            CALL ENTER(TEMPROB);
        END;
    END;
END;

ELSE IF (PROB_LOC(I) & "FF") = 4 THEN DO;
    TEMPROB = SUBSTR(PROBLEM(I),0,2);
    I1 = LOOKUP(TEMPROB);
    IF I1 = 0 THEN DO;
        CALL ENTER(TEMPROB);
```

```
            I2 = 0;
            DO J = SHR(PROB_LOC(I),8)+2 TO LENGTH(PROBLEM(I))-1;
               I2 = I2*10 + BYTE(SUBSTR(PROBLEM(I),J,1)) - "F0";
            END;
            PROB_TEMP = I2 - 1;
            OUTPUT1 = I2 + 1;
            CALL CODE4(TEMPROB, PROB_TEMP, OUTPUT1):
         END;
         ELSE CALL ERROR('IDENTIFIER HAS BEEN ENTERED IN THE SYMBOL TABLE');
      END;
   END;

   ELSE IF TYPE(I) = 1 THEN DO;

      IF (PROB_LOC(I) & "FF") = 1 THEN DO;
         TEMPROB = SUBSTR(PROBLEM(I),0,2);
         I1 = LOOKUP(TEMPROB);
         IF I1 = 0 THEN DO;
            CALL ENTER(TEMPROB);
            CALL CODE1(TEMPROB,SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+1));
         END;
         ELSE DO;
            PROB_TEMP = SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+1);
            I2 = LOOKUP(PROB_TEMP);
            IF I2 = 0 THEN DO;
               CALL ENTER(PROB_TEMP);
               CALL CODE1(PROB_TEMP,TEMPROB);
            END;
            ELSE CALL ERROR('BOTH IDENTIFIERS HAVE BEEN ENTERED IN THE ' ||
                  'SYMBOL TABLE',0);
         END;
      END;

      ELSE IF (PROB_LOC(I) & "FF") = 2 THEN DO;
         TEMPROB = SUBSTR(PROBLEM(I),0,2);
         I1 = LOOKUP(TEMPROB);
         IF I1 = 0 THEN DO;
            CALL ENTER(TEMPROB);
            PROB_TEMP = SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+1)||'-1';
            CALL CODE2(TEMPROB, PROB_TEMP);
         END;
         ELSE DO;
            PROB_TEMP = SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+1);
            I2 = LOOKUP(PROB_TEMP);
            IF I2 = 0 THEN DO;
               CALL ENTER(PROB_TEMP);
               TEMPROB = TEMPROB || '-1';
               CALL CODE2(PROB_TEMP, TEMPROB);
```

```
          END;
          ELSE CALL ERROR('BOTH IDENTIFIERS HAVE BEEN ENTERED IN THE ' ||
              'SYMBOL TABLE');
        END;
      END;

      ELSE IF (PROB_LOC(I) & "FF") = 3 THEN DO;
        TEMPROB = SUBSTR(PROBLEM(I),0,2);
        I1 = LOOKUP(TEMPROB);
        IF I1 = 0 THEN DO;
          CALL ENTER(TEMPROB);
          CALL CODE2(TEMPROB,SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+2));
        END;
        ELSE DO;
          PROB_TEMP = SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+2);
          I2 = LOOKUP(PROB_TEMP);
          IF I2 = 0 THEN DO;
            CALL ENTER(PROB_TEMP);
            CALL CODE2(PROB_TEMP, TEMPROB);
          END;
          ELSE CALL ERROR(' BOTH IDENTIFIERS HAVE BEEN ENTERED IN THE ' ||
              'SYMBOL TABLE',0);
        END;
      END;

      ELSE IF (PROB_LOC(I) & "FF") = 4 THEN DO;
        TEMPROB = SUBSTR(PROBLEM(I),0,2);
        PROB_TEMP = SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+2);
        I1 = LOOKUP(TEMPROB);
        IF I1 = 0 THEN DO;
          CALL ENTER(TEMPROB);
          OUTPUT1 = PROB_TEMP || '+1';
          PROB_TEMP = PROB_TEMP || '-1';
          CALL CODE4(TEMPROB, PROB_TEMP, OUTPUT1);
        END;
        ELSE DO;
          I1 = LOOKUP(PROB_TEMP);
          IF I1 = 0 THEN DO;
            CALL ENTER(PROB_TEMP);
            OUTPUT1 = TEMPROB || '+1';
            TEMPROB = TEMPROB || '-1';
            CALL CODE4(PROB_TEMP, TEMPROB, OUTPUT1);
          END;
          ELSE CALL ERROR('BOTH IDENTIFIERS HAVE BEEN ENTERED IN THE ' ||
            'SYMBOL TABLE',0);
        END;
      END;
    END;
```

```
ELSE IF TYPE(I) = 2 THEN DO;

   IF (PROB_LOC(I) & "FF") = 1 THEN DO;
      TEMPROB = SUBSTR(PROBLEM(I),0,2);
      PROB_TEMP = SUBSTR(PROBLEM(I),3,2);
      I1 = LOOKUP(TEMPROB);
      IF I1 = 0 THEN DO;
         CALL ENTER(TEMPROB);
         OUTPUT1 = SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+1)||'-'||PROB_TEMP
            ;
         CALL CODE3(TEMPROB,OUTPUT1);
      END;
      ELSE DO;
         I1 = LOOKUP(PROB_TEMP);
         IF I1 = 0 THEN DO;
            CALL ENTER(PROB_TEMP);
            OUTPUT1 = SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+1)||'-'||
               TEMPROB;
            CALL CODE3(PROB_TEMP,OUTPUT1);
         END;
         ELSE CALL ERROR('BOTH IDENTIFIERS HAVE BEEN ENTERED IN THE ' ||
            'SYMBOL TABLE',0);
      END;
   END;

   ELSE IF (PROB_LOC(I) & "FF") = 2 THEN DO;
      TEMPROB = SUBSTR(PROBLEM(I),0,2);
      IF BYTE(TEMPROB,0) > "F0" THEN DO;
         TEMPROB = SUBSTR(PROBLEM(I),0,SHR(PROB_LOC(I),8));
         PROB_TEMP = SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+1,2);
         OUTPUT1 = SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+4,2);
         I2 = 0 ;
         DO J = 0 TO LENGTH(TEMPROB)-1;
            I2 = I2 * 10 + BYTE(SUBSTR(TEMPROB,J,1)) - "F0";
         END;
         I2 = I2 + 1;
         I1 = LOOKUP(PROB_TEMP);
         IF I1 = 0 THEN DO;
            CALL ENTER(PROB_TEMP);
            TEMPROB = I2 || '-' || OUTPUT1;
            CALL CODE3(PROB_TEMP,TEMPROB);
         END;
         ELSE DO;
            I1 = LOOKUP(OUTPUT1);
            IF I1 = 0 THEN DO;
               CALL ENTER(OUTPUT1);
               TEMPROB = I2 || '-' || PROB_TEMP;
```

```
            CALL CODE3(OUTPUT1,TEMPROB);
         END;
         ELSE CALL ERROR('BOTH IDENTIFIERS HAVE BEEN ENTERED IN THE '
            || 'SYMBOL TABLE',0);
      END;
   END;
   ELSE DO;
      PROB_TEMP = SUBSTR(PROBLEM(I),3,2);
      I1 = LOOKUP(TEMPROB);
      IF I1 = 0 THEN DO;
         CALL ENTER(TEMPROB);
         OUTPUT1 = SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+1) || '-' ||
            PROB_TEMP;
         CALL CODE2(TEMPROB,OUTPUT1);
      END;
      ELSE DO;
         I1 = LOOKUP(PROB_TEMP);
         IF I1 = 0 THEN DO;
            CALL ENTER(PROB_TEMP);
            OUTPUT1 = SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+1) || '-' ||
               TEMPROB;
            CALL CODE2(PROB_TEMP,OUTPUT1);
         END;
         ELSE CALL ERROR('BOTH IDENTIFIERS HAVE BEEN ENTERED IN THE '||
            'SYMBOL TABLE',0);
      END;
   END;
END;

ELSE IF (PROB_LOC(I) & "FF") = 3 THEN DO;
   TEMPROB = SUBSTR(PROBLEM(I),0,2);
   IF BYTE(TEMPROB,0) > "F0" THEN DO;
      TEMPROB = SUBSTR(PROBLEM(I),0,SHR(PROB_LOC(I),8));
      PROB_TEMP = SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+2,2);
      OUTPUT1 = SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+5,2);
      I1 = LOOKUP(PROB_TEMP);
      IF I1 = 0 THEN DO;
         CALL ENTER(PROB_TEMP);
         TEMPROB = TEMPROB || '-' || OUTPUT1;
         CALL CODE3(OUTPUT1,TEMPROB);
      END;
      ELSE DO;
         I1 = LOOKUP(OUTPUT1);
         IF I1 = 0 THEN DO;
            CALL ENTER(OUTPUT1);
            TEMPROB = TEMPROB || '-' || PROB_TEMP;
            CALL CODE3(OUTPUT1,TEMPROB);
         END;
```

```
          ELSE CALL ERROR('BOTH IDENTIFIERS HAVE BEEN ENTERED IN THE '||
             'SYMBOL TABLE',0);
        END;
      END;
      ELSE DO;
        PROB_TEMP = SUBSTR(PROBLEM(I),3,2);
        I1 = LOOKUP(TEMPROB);
        IF I1 = 0 THEN DO:
          CALL ENTER(TEMPROB):
          OUTPUT1 = SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+2) || '-' ||
             PROB_TEMP;
          CALL CODE2(TEMPROB,OUTPUT1):
        END;
        ELSE DO;
          I1 = LOOKUP(PROB_TEMP);
          IF I1 = 0 THEN DO;
            CALL ENTER(PROB_TEMP);
            OUTPUT1 = SUBSTR(PROBLEM(I),SHR(PROB_LOC(I),8)+2) || '-' ||
               TEMPROB;
            CALL CODE2(PROB_TEMP,TEMPROB);
          END;
          ELSE CALL ERROR('BOTH IDENTIFIERS HAVE BEEN ENTERED IN THE '||
             'SYMBOL TABLE',0);
        END;
      END;
    END;
  END;
END;
DO I = 1 TO SY;
  L = LENGTH(SYMBOL(I));
  IF L <= 2 THEN GO TO FINIS;
  DO J = 0  TO L-2 BY 2:
    S = SUBSTR(SYMBOL(I),J,2);
    I1 = LOOKUP(S);
    IF I1 = 0 THEN DO;
      CALL ENTER(S);
      DATA_PTR = DATA_PTR + 1;
      DATA(DATA_PTR) = S || '=A_NUMBER(0->9)';
    END;
  END;
END;
FINIS: DO I = 1 TO AUX_PTR;
  DATA_PTR = DATA_PTR + 1 ;
  DATA(DATA_PTR) = AUX_DATA(I);
END;
DO I = 1 TO DATA_PTR;
  OUTPUT = DATA(I);
  OUTPUT(2) = DATA(I);
```

```
    END;
    OUTPUT(2) = DIRECTION || ';';
    OUTPUT = DIRECTION || ';';
    END RESOLVE;

DUMPIT:
    PROCEDURE;     /* DUMP OUT THE STATISTICS COLLECTED DURING THIS RUN  */
       DOUBLE_SPACE;
       /*  PUT OUT THE ENTRY COUNT FOR IMPORTANT PROCEDURES */

       OUTPUT = 'STACKING DECISIONS= ' || CALLCOUNT(1);
       OUTPUT = 'SCAN               = ' || CALLCOUNT(3);
       OUTPUT = 'FREE STRING AREA   = ' || FREELIMIT - FREEBASE;
    END DUMPIT;


STACK_DUMP:
    PROCEDURE;
       DECLARE LINE CHARACTER;
       LINE = 'PARTIAL PARSE TO THIS POINT IS: ';
       DO I = 2 TO SP;
          IF LENGTH(LINE) > 105 THEN
             DO;
                OUTPUT = LINE;
                LINE = X4;
             END;
          LINE = LINE || X1 || V(PARSE_STACK(I));
       END;
       OUTPUT = LINE;
    END STACK_DUMP;


  /*                    THE SYNTHESIS ALGORITHM FOR XPL                        */

SYNTHESIZE:
PROCEDURE(PRODUCTION_NUMBER);
    DECLARE (PRODUCTION_NUMBER, CONCNT) FIXED;
    DECLARE (ANS, QUES) BIT(1), CHAR CHARACTER;
    IF CONTROL(BYTE('P')) THEN CALL TRACER(PRODUCTION_NUMBER);

    DO CASE PRODUCTION_NUMBER;
      ;

 /*  <TEACHERS STATEMENT> ::= <TEACHERS DEFINITION>    */
    DO;
       IF MP ¬= 2 THEN  /* WE DIDN'T GET HERE LEGITIMATELY  */
          DO;
             CALL ERROR ('END; AT INVALID POINT', 1);
```

```
            CALL STACK_DUMP;
         END;
      OUTPUT(2) = 'EOF EOF EOF';
      COMPILING = FALSE;
   END;

/*  <TEACHERS DEFINITION> ::= <PROBLEM DEFINITION>     */
   DO;
      SAME: CALL RESOLVE;
      DO I = 1 TO DATA_PTR;
         AUX_DATA(I) = '';
         DATA(I) = '';
      END;
      DO I = 1 TO CONPTR;
         PROBLEM(I) = '';
         PROB_LOC(I), TYPE(I) = 0;
      END;
      DO I = 0 TO SY;
         SYMBOL(I) = '';
      END;
     SY, DATA_PTR, AUX_PTR = 0;
      CONPTR, CONCNT = 1;
      FLAG = TRUE;
   END;

/*  <TEACHERS DEFINITION> ::= <TEACHERS DEFINITION> <PROBLEM DEFINITION>     */
   GO TO SAME;

/*  <TEACHERS DEFINITION> ::= <FUNCTION DEFINITION>     */
   ;

/*  <TEACHERS DEFINITION> ::= <TEACHERS DEFINITION> <FUNCTION DEFINITION>     */
   ;

/*  <TEACHERS DEFINITION> ::= <DEFINITION DEFINITION>     */
   ;

/*  <TEACHERS DEFINITION> ::= <TEACHERS DEFINITION> <DEFINITION DEFINITION>
         */
   ;

/*  <PROBLEM DEFINITION> ::= <SET OPERATIONS>     */
   ;

/*  <PROBLEM DEFINITION> ::= <ARITHMETIC OPERATIONS> <PROBLEM CONSTRUCTION>
      <PROBLEM CONSTRAINTS>     */
   ABOVE: DIRECTION = VAR(MPP1);
```

```
/*  <PROBLEM DEFINITION> ::= <ARITHMETIC OPERATIONS> <PROBLEM CONSTRUCTION>
          */
  GO TC ABOVE:

/*  <PROBLEM DEFINITION> ::= <GEOMETRY>    */
  ;

/*  <PROBLEM DEFINITION> ::= <STATISTICS AND PROBABILITY>    */
  ;

/*  <PROBLEM DEFINITION> ::= <TRIGONOMETRY>    */
  ;

/*  <ARITHMETIC OPERATIONS> ::= <EXPRESSION> <RELATION> <EXPRESSION>    */
  DO;
     ANS, QUES = "0";
     DO I = 1 TO LENGTH(PROBLEM(1))-1;
        IF SUBSTR(PROBLEM(1),I,1) = '?' THEN QUES = "1";
        IF I <= LENGTH(PROBLEM(1))-6 THEN IF SUBSTR(PROBLEM(1),I,6) = 'ANSWER'
           THEN ANS = "1";
     END;
     IF NUM_ANS > 1 THEN DO;
        CALL ERROR(' MORE THAN ONE ANSWER IN THE SAME EQUATION ',1);
        RETURN;
     END;
     IF NUM_QUES > 1 THEN DO;
        CALL ERROR(' MORE THAN ONE ? IN THE SAME EQUATION ',1);
        RETURN;
     END;
     IF ANS & QUES THEN DO;
        CALL ERROR(' ANSWER AND QUESTION MARK ARE IN SAME EQUATION . ',1);
        RETURN;
     END;
     IF ¬ANS & ¬QUES THEN DO;
        CALL ERROR('NEITHER ANSWER NOR QUESTION MARK ARE IN THE EQUATION . ',1)
           ;
        RETURN;
     END;
     NUM_ANS, NUM_QUES = 0;
  END;

/*  <EXPRESSION> ::= <TERM>    */
  ;

/*  <EXPRESSION> ::= <EXPRESSION> + <TERM>    */
  ;

/*  <EXPRESSION> ::= <EXPRESSION> - <TERM>    */
```

```
  ;
/*  <EXPRESSION> ::= + <TERM>     */
  ;
/*  <EXPRESSION> ::= - <TERM>     */
  ;
/*  <TERM> ::= <PRIMARY>     */
  ;
/*  <TERM> ::= <TERM> * <PRIMARY>     */
  ;
/*  <TERM> ::= <TERM> / <PRIMARY>     */
  ;
/*  <PRIMARY> ::= <NUMBER>     */
  ;
/*  <PRIMARY> ::= <IDENTIFIER>     */
  DO;
     I1 = LOOKUP(VAR(MP));
     IF I1 = 0 THEN CALL ENTER(VAR(MP));
  END;
/*  <PRIMARY> ::= ( <ARITHMETIC OPERATIONS> )     */
  ;
/*  <PRIMARY> ::= ANSWER     */
  NUM_ANS = NUM_ANS + 1;
/*  <RELATION> ::= =     */
  ;
/*  <RELATION> ::= <     */
  ;
/*  <RELATION> ::= >     */
  ;
/*  <RELATION> ::= ¬ =     */
  ;
/*  <RELATION> ::= ¬ <     */
  ;
/*  <RELATION> ::= ¬ >     */
```

```
  ;
/*  <RELATION> ::= < =    */
  ;
/*  <RELATION> ::= > =    */
  ;
/*  <RELATION> ::= ?    */
  NUM_QUES = NUM_QUES + 1;
/*  <PROBLEM CONSTRUCTION> ::= <PROBLEM DIRECTION> CARRIES IN <IDENTIFIER>
    PLACE    */
  DO;
     IF VAR(SP-1) = 'ONES' THEN POSITION = 1; ELSE
     IF VAR(SP-1) = 'TENS' THEN POSITION = 2; ELSE
     IF VAR(SP-1) = 'HUNDREDS' THEN POSITION = 3; ELSE
     IF VAR(SP-1) = 'THOUSANDS' THEN POSITION = 4;
     STA: I1 = SY;
     CHAR = '';
     DO J = 1 TO I1;
        IF SYMBOL(J) ¬= 'ANSWER' THEN
           IF CHAR = '' THEN DO:
              CHAR = SUBSTR(SYMBOL(J),0,1) || POSITION;
              CALL ENTER(CHAR);
           END;
           ELSE DO;
              CALL CODE3(CHAR, SUBSTR(SYMBOL(J),0,1) || POSITION);
              GO TO CONT;
           END;
     END:
  CONT: END;
/*  <PROBLEM CONSTRUCTION> ::= <PROBLEM DIRECTION> CARRIES IN POSITION <NUMBER>
       */
  DO;
     POSITION = FIXV(SP);
     GO TO STA;
  END;
/*  <PROBLEM CONSTRUCTION> ::= <PROBLEM DIRECTION> BARROWS IN <IDENTIFIER>
    PLACE    */
  DO;
     IF VAR(SP-1) = 'ONES' THEN POSITION = 1; ELSE
     IF VAR(SP-1) = 'TENS' THEN POSITION = 2; ELSE
     IF VAR(SP-1) = 'HUNDREDS' THEN POSITION = 3; ELSE
     IF VAR(SP-1) = 'THOUSANDS' THEN POSITION = 4;
  END;
```

```
/*  <PROBLEM CONSTRUCTION> ::= <PROBLEM DIRECTION> BARROWS IN POSITION <NUMBER>
        */
     POSITION = FIXV(SP);

/*  <PROBLEM CONSTRUCTION> ::= <PROBLEM DIRECTION>     */
  POSITION = 0;

/*  <PROBLEM DIRECTION> ::= VERT     */
  ;

/*  <PROBLEM DIRECTION> ::= VERTICAL     */
  VAR(MP) = 'VERT';

/*  <PROBLEM DIRECTION> ::= HORIZ     */
  VAR(MP) = 'HOR';

/*  <PROBLEM DIRECTION> ::= HOR    .*/
  ;

/*  <PROBLEM DIRECTION> ::= HORIZONAL     */
  VAR(MP) = 'HOR';

/*  <PROBLEM CONSTRAINTS> ::= <PROBLEM CONSTRAINTS RIGHT>     */
  ;

/*  <PROBLEM CONSTRAINTS> ::= <PROBLEM CONSTRAINTS> <PROBLEM CONSTRAINTS RIGHT>
        */
  ;

/*  <PROBLEM CONSTRAINTS RIGHT> ::= <LEFT PROBLEM CONSTRAINT> <NUMBER>     */
  DO;
     CONCNT = CONCNT + 1;
     IF FIXV(MP) = 3 THEN TYPE(CONCNT) = 2; ELSE
     IF FIXV(MP) = 1 THEN TYPE(CONCNT) = 0;
  END;

/*  <PROBLEM CONSTRAINTS RIGHT> ::= <LEFT PROBLEM CONSTRAINT> <IDENTIFIER>
        */
  DO;
     CONCNT = CONCNT + 1;
     TYPE(CONCNT) = FIXV(MP);
     I2 = LENGTH(VAR(SP));
     CALL FIND(VAR(SP));
     IF I3 = 0 THEN CALL ERROR(' INVALID CONSTRAINT ',1);
  END;

/*  <LEFT PROBLEM CONSTRAINT> ::= <MIDDLE CONSTRAINT> <IDENTIFIER> <RELATION>
```

```
          */
   DO;
      FIXV(MP) = 3;
      GO TO MID;
   END;

/*  <LEFT PROBLEM CONSTRAINT> ::= , <IDENTIFIER> <RELATION>     */
   DO;
      FIXV(MP) = 1;
      MID: I2 = LENGTH(VAR(MPP1));
      I2 = LENGTH(VAR(MPP1));
      CALL FIND(VAR(MPP1),I2);
      IF I2 = C THEN DO;
         CALL ERROR(' INVALID CONSTRAINTS ',1);
      END;
   END;

/*  <MIDDLE CONSTRAINT> ::= , <IDENTIFIER> +     */
   GO TO MID;

/*  <SET OPERATIONS> ::= <UNION>     */
   ;

/*  <SET OPERATIONS> ::= <INTERSECTION>     */
   ;

/*  <SET OPERATIONS> ::= <COMPLEMENT>     */
   ;

/*  <SET OPERATIONS> ::= <DISJOINT>     */
   ;

/*  <SET OPERATIONS> ::= <EQUAL>     */
   ;

/*  <GEOMETRY> ::= <METRIC>     */
   ;

/*  <GEOMETRY> ::= <NON-METRIC>     */
   ;

/*  <STATISTICS AND PROBABILITY> ::= <MEASURE OF CENTRAL TENDENCY>     */
   ;

/*  <METRIC> ::= <AREA OF REGIONS>     */
   ;

/*  <NON-METRIC> ::= <LINES, POINTS AND CURVES>     */
```

```
   ;
 /*  <NON-METRIC> ::= <PLANES>     */
   ;
 /*  <NON-METRIC> ::= <ANGLES>     */
   ;
 /*  <NON-METRIC> ::= <POLYGONS>     */
   ;
 /*  <NON-METRIC> ::= <CIRCLES>     */
   ;
 /*  <NON-METRIC> ::= <CONSTRUCTION>     */
   ;
 /*  <MEASURE OF CENTRAL TENDENCY> ::= <AVERAGE>     */
   ;
 /*  <MEASURE OF CENTRAL TENDENCY> ::= <RANGE>     */
   ;
 /*  <MEASURE OF CENTRAL TENDENCY> ::= <MODE>     */
   ;
 /*  <MEASURE OF CENTRAL TENDENCY> ::= <MEDIAN>     */
   ;
   END;
END SYNTHESIZE;

  /*                 SYNTACTIC PARSING FUNCTIONS                                   */

RIGHT_CONFLICT:
   PROCEDURE (LEFT) BIT(1);
      DECLARE LEFT FIXED;
      /*  THIS PROCEDURE IS TRUE IF TOKEN IS ¬ A LEGAL RIGHT CONTEXT OF LEFT*/
      RETURN ("C0" & SHL(BYTE(C1(LEFT), SHR(TOKEN,2)), SHL(TOKEN,1)
         & "(1)6")) = 0;
   END RIGHT_CONFLICT;

RECOVER:
   PROCEDURE;
      /* IF THIS IS THE SECOND SUCCESSIVE CALL TO RECOVER, DISCARD ONE SYMBOL */
      IF ¬ FAILSOFT THEN CALL SCAN;
```

```
         FAILSOFT = FALSE;
         DO WHILE ¬ STOPIT(TOKEN);
            CALL SCAN;  /* TO FIND SOMETHING SOLID IN THE TEXT  */
         END;
         DO WHILE RIGHT_CONFLICT (PARSE_STACK(SP));
            IF SP > 2 THEN SP = SP - 1;   /* AND IN THE STACK  */
            ELSE CALL SCAN;  /* BUT DON'T GO TOO FAR  */
         END;
         OUTPUT = 'RESUME:' || SUBSTR(POINTER, TEXT_LIMIT-CP+MARGIN_CHOP+7);
      END RECOVER;

   STACKING:
      PROCEDURE BIT(1);  /* STACKING DECISION FUNCTION */
         CALLCOUNT(1) = CALLCOUNT(1) + 1;
         DO FOREVER;    /* UNTIL RETURN  */
            DO CASE SHR(BYTE(C1(PARSE_STACK(SP)),SHR(TOKEN,2)),SHL(3-TOKEN,1)&6)&3;

               /*  CASE 0  */
               DO;    /* ILLEGAL SYMBOL PAIR  */
                  CALL ERROR('ILLEGAL SYMBOL PAIR: ' || V(PARSE_STACK(SP)) || X1 ||
                     V(TOKEN), 1);
                  CALL STACK_DUMP;
                  CALL RECOVER;
               END;

               /*  CASE 1  */

               RETURN TRUE;       /*  STACK TOKEN  */

               /*  CASE 2  */

               RETURN FALSE;      /* DON'T STACK IT YET  */

               /*  CASE 3  */

               DO;      /* MUST CHECK TRIPLES  */
                  J = SHL(PARSE_STACK(SP-1), 16) + SHL(PARSE_STACK(SP), 8) + TOKEN;
                  I = -1;  K = NC1TRIPLES + 1;  /* BINARY SEARCH OF TRIPLES  */
                  DO WHILE I + 1 < K;
                     L = SHR(I+K, 1);
                     IF C1TRIPLES(L) > J THEN K = L;
                     ELSE IF C1TRIPLES(L) < J THEN I = L;
                     ELSE RETURN TRUE;  /* IT IS A VALID TRIPLE  */
                  END;
                  RETURN FALSE;
               END;

            END;    /* OF DO CASE  */
```

```
        END;   /*  CF DO FOREVER */
    END STACKING;

  PR_CK:
    PROCEDURE(PRD) BIT(1);
      /* DECISION PROCEDURE FOR CONTEXT CHECK OF EQUAL OR IMBEDDED RIGHT PARTS*/
      DECLARE (H, I, J, PRD) FIXED;
      DO CASE CONTEXT_CASE(PRD);

         /*  CASE 0 -- NO CHECK REQUIRED  */

         RETURN TRUE;

         /*  CASE 1 -- RIGHT CONTEXT CHECK  */

         RETURN ¬ RIGHT_CONFLICT (HDTB(PRD));

         /*  CASE 2 -- LEFT CONTEXT CHECK  */

         DO;
            H = HDTB(PRD) - NT;
            I = PARSE_STACK(SP - PRLENGTH(PRD));
            DO J = LEFT_INDEX(H-1) TO LEFT_INDEX(H) - 1;
               IF LEFT_CONTEXT(J) = I THEN RETURN TRUE;
            END;
            RETURN FALSE;
         END;

         /*  CASE 3 -- CHECK TRIPLES  */

         DO;
            H = HDTB(PRD) - NT;
            I = SHL(PARSE_STACK(SP - PRLENGTH(PRD)), 8) + TOKEN;
            DO J = TRIPLE_INDEX(H-1) TO TRIPLE_INDEX(H) - 1;
               IF CONTEXT_TRIPLE(J) = I THEN RETURN TRUE;
            END;
            RETURN FALSE;
         END;

      END;  /* OF DO CASE  */
    END PR_OK;

   /*                   ANALYSIS ALGORITHM                                    */

  REDUCE:
    PROCEDURE;
      DECLARE (I, J, PRD) FIXED;
```

```
        /* PACK STACK TOP INTO ONE WORD */
        DO I = SP - 4 TO SP - 1;
           J = SHL(J, 8) + PARSE_STACK(I);
        END;

        DO PRD = PR_INDEX(PARSE_STACK(SP)-1) TO PR_INDEX(PARSE_STACK(SP)) - 1;
           IF (PRMASK(PRLENGTH(PRD)) & J) = PRTB(PRD) THEN
              IF PR_OK(PRD) THEN
              DO;  7* AN ALLOWED REDUCTION */
                 MP = SP - PRLENGTH(PRD) + 1; MPP1 = MP + 1;
                 CALL SYNTHESIZE(PRDTB(PRD));
                 SP = MP;
                 PARSE_STACK(SP) = HDTB(PRD);
                 RETURN;
              END;
        END;

        /* LOOK UP HAS FAILED, ERROR CONDITION */
        CALL ERROR('NO PRODUCTION IS APPLICABLE',1);
        CALL STACK_DUMP;
        FAILSOFT = FALSE;
        CALL RECOVER;
     END REDUCE;


  COMPILATION_LOOP:
     PROCEDURE:

        COMPILING = TRUE;
        FLAG = TRUE:
        CONPTR = 1:
        DO I = 1 TO 32;
           PROBLEM(I) = '';
        END;
        DO WHILE COMPILING;     /* ONCE AROUND FOR EACH PRODUCTION (REDUCTION)  */
           DO WHILE STACKING;
              SP = SP + 1;
              IF SP = STACKSIZE THEN
                 DO;
                    CALL ERROR ('STACK OVERFLOW *** CHECKING ABORTED ***', 2);
                    RETURN;   /* THUS ABORTING CHECKING */
                 END;
              PARSE_STACK(SP) = TOKEN;
              VAR(SP) = BCD;
              FIXV(SP) = NUMBER_VALUE;
              IF SUBSTR(BCD,0,3) = 'VER' | SUBSTR(BCD,0,3) = 'HOR'
                 THEN FLAG = FALSE;
              IF FLAG THEN PROBLEM(CONPTR) = PROBLEM(CONPTR) || VAR(SP);
```

```
            CALL SCAN;
         END;

         CALL REDUCE;
      END;      /* OF DO WHILE COMPILING  */
   END COMPILATION_LOOP;


PRINT_SUMMARY:
   PROCEDURE;
      DECLARE I FIXED;
      CALL PRINT_DATE_AND_TIME ('END OF CHECKING ', DATE, TIME);
      OUTPUT = '';
      OUTPUT = CARD_COUNT || ' CARDS WERE CHECKED.';
      IF ERROR_COUNT = 0 THEN OUTPUT = 'NO ERRORS WERE DETECTED.';
      ELSE IF ERROR_COUNT > 1 THEN
         OUTPUT = ERROR_COUNT || ' ERRORS (' || SEVERE_ERRORS
            || ' SEVERE) WERE DETECTED.';
      ELSE IF SEVERE_ERRORS = 1 THEN OUTPUT = 'ONE SEVERE ERROR WAS DETECTED.';
         ELSE OUTPUT = 'ONE ERROR WAS DETECTED.';
      IF PREVIOUS_ERROR > 0 THEN
         OUTPUT = 'THE LAST DETECTED ERROR WAS ON LINE ' || PREVIOUS_ERROR
            || PERIOD;
      IF CONTROL(BYTE('D')) THEN CALL DUMPIT;
      DOUBLE_SPACE;
      CLOCK(3) = TIME;
      DO I = 1 TO 3;    /* WATCH OUT FOR MIDNIGHT */
         IF CLOCK(I) < CLOCK(I-1) THEN CLOCK(I) = CLOCK(I) +  8640000;
      END;
      CALL PRINT_TIME ('TOTAL TIME IN CHECKER     ', CLOCK(3) - CLOCK(0));
      CALL PRINT_TIME ('SET UP TIME               ', CLOCK(1) - CLOCK(0));
      CALL PRINT_TIME ('ACTUAL CHECKING TIME      ', CLOCK(2) - CLOCK(1));
      CALL PRINT_TIME ('CLEAN-UP TIME AT END      ', CLOCK(3) - CLOCK(2));
      IF CLOCK(2) > CLOCK(1) THEN   /* WATCH OUT FOR CLOCK BEING OFF */
      OUTPUT = 'CHECKING RATE: ' || 6000*CARD_COUNT/(CLOCK(2)-CLOCK(1))
         || ' CARDS PER MINUTE.';
   END PRINT_SUMMARY;


MAIN_PROCEDURE:
   PROCEDURE;
      CLOCK(0) = TIME;  /* KEEP TRACK OF TIME IN EXECUTION */
      CALL INITIALIZATION;

      CLOCK(1) = TIME;

      CALL COMPILATION_LOOP;
```

```
        CLOCK(2) = TIME;

        /* CLOCK(3) GETS SET IN PRINT_SUMMARY */
        CALL PRINT_SUMMARY;

     END MAIN_PROCEDURE;

  CALL MAIN_PROCEDURE;
  RETURN SEVERE_ERRORS;

  EOF EOF EOF
```

APPENDIX C

Format description of language for communicating with PROBLEM DESCRIPTION INTERPRETER.

The meta-symbols used in the description of the language serve the following functions:

```
::=  is used to indicate a definition

|    is used to indicate alternate definitions

< >  are used to enclose items which are elements of
     the meta-language which describe the elements of
     the PROBLEM DESCRIPTION INTERPRETER language
```

```
<interpreter>  ::=  <interpreter statement> ;

<interpreter statement> ::= <problem statement> | <interpreter
                            statement> ; <problem statement>
                            | <interpreter statement> <right
                            statement> | <interpreter
                            statement> <direction>

<right statement>  ::= <left part> <a_number(>

<a_number(> ::=  <a_number> <digit part>

<a_number>  ::=  A_NUMBER (<identifier> | A_NUMBER(
                 <number>

<digit part> ::=  ) | <minus identifier> <arrow number> ) |
                  <right part> <plus numbe > <arrow number> )
                  | <arrow number <minus identifier> ) |
                  <arrow identifier> <minus number> ) |
                  <arrow identifier> ) | <arrow number> )

<arrow number>  ::=  -> <number>

<arrow identifier>  ::=  -> <identifier>

<right part>  ::= <arrow identifier> <minus number>  ,
                  <identifier>

<plus number>  ::=  + <number>

<minus number>  ::=  - <number>

<left part>  ::=  <identifier =>
```

```
<identifier =>  ::=  <identifier> =

<problem statement> ::= <expression> <relation> <expression>

<expression> ::= <term> | <expression> + <term> | <expression>
                 - <term>

<term> ::= <primary> | <term> * <primary> | <term> / <primary>

<primary> ::= <number> | <identifier> | ( <expression> ) |
              ANSWER

<relation> ::= = | < | > | ≠ |  ≮  |  ≯  | < = | > = | ?

<direction> ::=  HOR | VERT
```

```
                        APPENDIX  D


           PROBLEM DESCRIPTION INTERPRETER

                    R. J. WOOLS

                 COMPUTER SCIENCE
                      STUDENT

                NAVAL POSTGRADUATE
                      SCHOOL

                    CALIFORNA
                      93940

   */

DECLARE
   MAXSYM         LITERALLY      '64';
DECLARE
   (I1, I2, I3, I4) FIXED,
   SYMBOL(MAXSYM) CHARACTER,
   VALUE_OF(MAXSYM) FIXED,
   NUMBER_OF(MAXSYM) CHARACTER,
   FLAG(MAXSYM) BIT(1),
   VAL_FLAG(MAXSYM) BIT(1),
   NUMBER_STRING CHARACTER,
   SY FIXED,
   PROBLEM CHARACTER,
   PROB_FLAG BIT(1),
   (SY_CNT, SY_LINE) FIXED;


DECLARE NSY LITERALLY '43', NT LITERALLY '20';
DECLARE V(NSY) CHARACTER INITIAL ( '<ERROR: TOKEN = 0>', ';', '(', ')', '-',
   '>', ',', '+', '=', '*', '/', '<', '¬', '?', '_|_', 'HOR', 'VERT',
   'ANSWER', 'A_NUMBER', '<NUMBER>', '<IDENTIFIER>', '<+>', '<->', '<(2>',
   '<TERM>', '<PRIMARY>', '<A_NUMBER>', '<FELATION>', '<DIRECTION>',
   '<LEFT PART>', '<A_NUMBER(>', '<INTERPUTER>', '<DIGIT PART>',
   '<RIGHT PART>', '<EXPRESSION>', '<PLUS NUMBER>', '<IDENTIFIER =>',
   '<ARROW NUMBER>', '<MINUS NUMBEF>', '<RIGHT STATEMENT>',
   '<MINUS IDENTIFIER>', '<ARROW IDENTIFIEF>', '<PROBLEM STATEMENT>',
   '<INTERPUTER STATEMENT>');
DECLARE V_INDEX(12) BIT(8) INITIAL ( 1, 14, 14, 16, 17, 17, 18, 18, 20, 20,
   20, 20, 21);
```

```
DECLARE C1(NSY) BIT(42) INITIAL (
    "(2) 00000 00000 00000 00000 0",
    "(2) 00100 00000 00002 00101 1",
    "(2) 00200 00000 00000 00203 3",
    "(2) 02022 20222 22220 22000 2",
    "(2) 00200 10000 00000 00203 3",
    "(2) 00200 00010 00000 00203 3",
    "(2) 00000 00000 00000 00000 1",
    "(2) 00200 00000 00000 00203 2",
    "(2) 00200 00000 00000 00222 2",
    "(2) 00100 00000 00000 00101 1",
    "(2) 00100 00000 00000 00101 1",
    "(2) 00200 00010 00000 00202 2",
    "(2) 00000 10010 01000 00000 0",
    "(2) 00200 00000 00000 00202 2",
    "(2) 00100 00000 00000 00101 1",
    "(2) 02000 00000 00000 22000 2",
    "(2) 02000 00000 00000 22000 2",
    "(2) 02022 20222 22220 22000 2",
    "(2) 00100 00000 00000 00000 0",
    "(2) 02022 22222 22220 22000 2",
    "(2) 02022 20232 22220 22000 2",
    "(2) 00100 00000 00000 00101 1",
    "(2) 00100 00000 00000 00101 1",
    "(2) 00100 00000 00000 00101 1",
    "(2) 02022 20221 12220 22000 2",
    "(2) 02022 20222 22220 22000 2",
    "(2) 02000 00000 00000 22000 2",
    "(2) 00100 00000 00000 00101 1",
    "(2) 02000 00000 00000 22000 2",
    "(2) 00000 00000 00000 00010 0",
    "(2) 00011 00000 00000 00000 0",
    "(2) 00000 00000 00002 00000 0",
    "(2) 02000 00000 00000 22000 2",
    "(2) 00000 00100 00000 00000 0",
    "(2) 02011 10110 01110 22000 2",
    "(2) 00001 00000 00000 00000 0",
    "(2) 00000 00000 00000 00020 0",
    "(2) 00011 00000 00000 00000 0",
    "(2) 00010 01000 00000 00000 0",
    "(2) 02000 00000 00000 22000 2",
    "(2) 00011 00000 00000 00000 0",
    "(2) 00011 00000 00000 00000 0",
    "(2) 02000 00000 00000 22000 2",
    "(2) 01000 00000 00000 11000 1");
DECLARE NC1TRIPLES LITERALLY '8';
DECLARE C1TRIPLES(NC1TRIPLES) FIXED INITIAL ( 263443, 263444, 1180179,
    1180180, 1967124, 2164499, 2425876, 2688019, 2823176);
```

```
DECLARE PRTB(49) FIXED INITIAL (0, 43, 0, 2171685, 10277, 9512, 10534, 5922,
   41, 37, 0, 0, 12, 0, 0, 20, 12, 11, 5, 0, 12, 0, 0, 0, 0, 0, 4610, 1029,
   7, 4, 0, 2696710, 4610, 1029, 4, 0, 8725, 8726, 0, 6153, 6154, 0, 29, 43,
   30, 8731, 0, 43, 11009, 0);
DECLARE PRDTB(49) BIT(8) INITIAL (0, 1, 36, 13, 12, 14, 15, 34, 16, 17, 11,
   37, 44, 41, 38, 24, 42, 45, 46, 39, 43, 40, 47, 48, 49, 35, 10, 18, 21,
   22, 32, 20, 9, 19, 23, 33, 27, 28, 26, 30, 31, 29, 6, 5, 8, 25, 7, 4, 3,
   2);
DECLARE HDTB(49) BIT(8) INITIAL (0, 31, 23, 32, 32, 32, 32, 25, 32, 32, 32,
   22, 27, 27, 21, 36, 27, 27, 27, 27, 27, 27, 27, 28, 28, 25, 30, 37, 35,
   38, 25, 33, 30, 41, 40, 25, 34, 34, 34, 24, 24, 24, 39, 43, 26, 42, 29,
   43, 43, 43);
DECLARE PRLENGTH(49) BIT(8) INITIAL (0, 2, 1, 4, 3, 3, 3, 3, 2, 2, 1, 1, 2,
   1, 1, 2, 2, 2, 2, 1, 2, 1, 1, 1, 1, 1, 3, 3, 2, 2, 1, 4, 3, 3, 2, 1, 3, 3,
   1, 3, 3, 1, 2, 2, 2, 3, 1, 2, 3, 1);
DECLARE CONTEXT_CASE(49) BIT(8) INITIAL (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
DECLARE LEFT_CONTEXT(0) BIT(8) INITIAL ( 0);
DECLARE LEFT_INDEX(23) BIT(8) INITIAL ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1);
DECLARE CONTEXT_TRIPLE(0) FIXED INITIAL ( 0);
DECLARE TRIPLE_INDEX(23) BIT(8) INITIAL ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1);
DECLARE PR_INDEX(43) BIT(8) INITIAL ( 1, 2, 3, 11, 12, 14, 14, 15, 20, 20,
   20, 22, 22, 23, 23, 24, 25, 26, 26, 31, 36, 36, 36, 36, 39, 42, 43, 43,
   44, 44, 44, 44, 45, 45, 46, 46, 47, 47, 47, 48, 48, 48, 50, 50);

/*  END OF CARDS PUNCHED BY SYNTAX                                          */

/*  DECLARATIONS FOR THE SCANNER                                            */

/* TOKEN IS THE INDEX INTO THE VOCABULARY V() OF THE LAST SYMBOL SCANNED,
   CP IS THE POINTER TO THE LAST CHARACTER SCANNED IN THE CARDIMAGE,
   BCD IS THE LAST SYMBOL SCANNED (LITERAL CHARACTER STRING). */
DECLARE (TOKEN, CP) FIXED, BCD CHARACTER;

/* SET UP SOME CONVENIENT ABBREVIATIONS FOR PRINTER CONTROL */
DECLARE EJECT_PAGE LITERALLY 'OUTPUT(1) = PAGE',
   PAGE CHARACTER INITIAL ('1'), DOUBLE CHARACTER INITIAL ('0'),
   DOUBLE_SPACE LITERALLY 'OUTPUT(1) = DOUBLE',
   X70 CHARACTER INITIAL ('
               ');

/* LENGTH OF LONGEST SYMBOL IN V */
DECLARE (RESERVED_LIMIT, MARGIN_CHOP) FIXED;

/* CHARTYPE() IS USED TO DISTINGUISH CLASSES OF SYMBOLS IN THE SCANNER.
```

```
   TX() IS A TABLE USED FOR TRANSLATING FROM ONE CHARACTER SET TO ANOTHER.
   CONTROL() HOLDS THE VALUE OF THE COMPILER CONTROL TOGGLES SET IN $ CARDS.
   NOT_LETTER_OR_DIGIT() IS SIMILIAR TO CHARTYPE() BUT USED IN SCANNING
   IDENTIFIERS ONLY.

   ALL ARE USED BY THE SCANNER AND CONTROL() IS SET THERE.
*/
DECLARE (CHARTYPE, TX) (255) BIT(8),
        (CONTROL, NOT_LETTER_OR_DIGIT)(255) BIT(1);

/* ALPHABET CONSISTS OF THE SYMBOLS CONSIDERED ALPHABETIC IN BUILDING
   IDENTIFIERS     */
DECLARE ALPHABET CHARACTER INITIAL ('ABCDEFGHIJKLMNOPQRSTUVWXYZ_$@#');

/* BUFFER HOLDS THE LATEST CARDIMAGE,
   TEXT HOLDS THE PRESENT STATE OF THE INPUT TEXT
   (NOT INCLUDING THE PORTIONS DELETED BY THE SCANNER),
   TEXT_LIMIT IS A CONVENIENT PLACE TO STORE THE POINTER TO THE END OF TEXT,
   CARD_COUNT IS INCREMENTED BY ONE FOR EVERY SOURCE CARD READ,
   ERROR_COUNT TABULATES THE ERRORS AS THEY ARE DETECTED,
   SEVERE_ERRORS TABULATES THOSE ERRORS OF FATAL SIGNIFICANCE.
*/
DECLARE (BUFFER, TEXT) CHARACTER,
   (TEXT_LIMIT, CARD_COUNT, ERROR_COUNT, SEVERE_ERRORS, PREVIOUS_ERROR) FIXED
   ;

/* NUMBER_VALUE CONTAINS THE NUMERIC VALUE OF THE LAST CONSTANT SCANNED,
*/
DECLARE NUMBER_VALUE FIXED;

/* EACH OF THE FOLLOWING CONTAINS THE INDEX INTO V() OF THE CORRESPONDING
   SYMBOL.   WE ASK:    IF TOKEN = IDENT     ETC.    */
DECLARE (IDENT, NUMBER, DIVIDE, EOFILE) FIXED;

/* STOPIT() IS A TABLE OF SYMBOLS WHICH ARE ALLOWED TO TERMINATE THE ERROR
   FLUSH PROCESS.  IN GENERAL THEY ARE SYMBOLS OF SUFFICIENT SYNTACTIC
   HIERARCHY THAT WE EXPECT TO AVOID ATTEMPTING TO START CHECKING AGAIN
   RIGHT INTO ANOTHER ERROR PRODUCING SITUATION.  THE TOKEN STACK IS ALSO
   FLUSHED DOWN TO SOMETHING ACCEPTABLE TO A STOPIT() SYMBOL.
   FAILSOFT IS A BIT WHICH ALLOWS THE COMPILER ONE ATTEMPT AT A GENTLE
   RECOVERY.   THEN IT TAKES A STRONG HAND.   WHEN THERE IS REAL TROUBLE
   COMPILING IS SET TO FALSE, THEREBY TERMINATING THE COMPILATION.
*/
DECLARE STOPIT(100) BIT(1), (FAILSOFT, COMPILING) BIT(1);

DECLARE S CHARACTER;  /* A TEMPORARY USED VARIOUS PLACES */

/* THE ENTRIES IN PRMASK() ARE USED TO SELECT OUT PORTIONS OF CODED
```

```
   PRODUCTIONS AND THE STACK TOP FOR COMPARISON IN THE ANALYSIS ALGORITHM */
DECLARE PRMASK(5) FIXED INITIAL (0, 0, "FF", "FFFF", "FFFFFF", "FFFFFFFF");

/*THE PROPER SUBSTRING OF POINTER IS USED TO PLACE AN  |  UNDER THE POINT
   OF DETECTION OF AN ERROR DURING CHECKING.  IT MARKS THE LAST CHARACTER
   SCANNED.  */
DECLARE POINTER CHARACTER INITIAL ('
                                               |');
DECLARE CALLCOUNT(20) FIXED   /* COUNT THE CALLS OF IMPORTANT PROCEDURES */
   INITIAL(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);

/* RECORD THE TIMES OF IMPORTANT POINTS DURING CHECKING */
DECLARE CLOCK(5) FIXED;


/* COMMONLY USED STRINGS */
DECLARE X1 CHARACTER INITIAL(' '), X4 CHARACTER INITIAL('    ');
DECLARE PERIOD CHARACTER INITIAL ('.');

/* TEMPORARIES USED THROUGHOUT THE COMPILER */
DECLARE (I, J, K, L) FIXED;

DECLARE TRUE LITERALLY '1', FALSE LITERALLY '0', FOREVER LITERALLY 'WHILE 1';

/*  THE STACKS DECLARED BELOW ARE USED TO DRIVE THE SYNTACTIC
   ANALYSIS ALGORITHM AND STORE INFORMATION RELEVANT TO THE INTERPRETATION
   OF THE TEXT.  THE STACKS ARE ALL POINTED TO BY THE STACK POINTER SP.  */

DECLARE STACKSIZE LITERALLY '75';  /* SIZE OF STACK  */
DECLARE PARSE_STACK (STACKSIZE) BIT(8); /* TOKENS OF THE PARTIALLY PARSED
                                           TEXT */
DECLARE VAR (STACKSIZE) CHARACTER;/* EBCDIC NAME OF ITEM */
DECLARE FIXV (STACKSIZE) FIXED;    /* FIXED (NUMERIC) VALUE */

/* SP POINTS TO THE RIGHT END OF THE REDUCIBLE STRING IN THE PARSE STACK,
   MP POINTS TO THE LEFT END, AND
   MPP1 = MP+1.
*/
DECLARE (SP, MP, MPP1) FIXED;

/*                  P R O C E D U R E S                                   */

PAD:
   PROCEDURE (STRING, WIDTH) CHARACTER;
      DECLARE STRING CHARACTER, (WIDTH, L) FIXED;

      L = LENGTH(STRING);
```

```
        IF L >= WIDTH THEN RETURN STRING;
        ELSE RETURN STRING || SUBSTR(X70, 0, WIDTH-L);
     END PAD;

  C_FORMAT: PROCEDURE(STRING, WIDTH) CHARACTER;
     DECLARE (WIDTH, L) FIXED, STRING CHARACTER;
     L = LENGTH(STRING);
     IF L >= WIDTH THEN RETURN STRING;
     ELSE RETURN SUBSTR(X70, 0, WIDTH-L) || STRING;
     END C_FORMAT;

  I_FORMAT:
     PROCEDURE (NUMBER, WIDTH) CHARACTER;
        DECLARE (NUMBER, WIDTH, L) FIXED, STRING CHARACTER;

        STRING = NUMBER;
        L = LENGTH(STRING);
        IF L >= WIDTH THEN RETURN STRING;
        ELSE RETURN SUBSTR(X70, 0, WIDTH-L) || STRING;
     END I_FORMAT;

  ERROR:
     PROCEDURE(MSG, SEVERITY);
        /* PRINTS AND ACCOUNTS FOR ALL ERROR MESSAGES */
        /* IF SEVERITY IS NOT SUPPLIED, 0 IS ASSUMED */
        DECLARE MSG CHARACTER, SEVERITY FIXED;
        ERROR_COUNT = ERROR_COUNT + 1;
        /* IF LISTING IS SUPPRESSED, FORCE PRINTING OF THIS LINE */
        IF ¬ CONTROL(BYTE('L')) THEN
           OUTPUT = I_FORMAT (CARD_COUNT, 4) || ' |' || BUFFER || '|';
        OUTPUT = SUBSTR(POINTER, TEXT_LIMIT-CP+MARGIN_CHOP);
        OUTPUT = '*** ERROR, ' || MSG ||
             '. LAST PREVIOUS ERROR WAS DETECTED ON LINE ' ||
             PREVIOUS_ERROR || '.  ***';
        PREVIOUS_ERROR = CARD_COUNT;
        IF SEVERITY > 0 THEN
           IF SEVERE_ERRORS > 25 THEN
             DO;
                OUTPUT = '*** TOO MANY SEVERE ERRORS, CHECKING ABORTED ***';
                COMPILING = FALSE;
             END;
           ELSE SEVERE_ERRORS = SEVERE_ERRORS + 1;
     END ERROR;

   /*                     CARD IMAGE HANDLING PROCEDURE                      */

  GET_CARD:
     PROCEDURE;
```

```
      /* DOES ALL CARD READING AND LISTING                                      */
      DECLARE I FIXED, (TEMP, TEMP0, REST) CHARACTER, READING BIT(1);
            BUFFER = INPUT;
            IF LENGTH(BUFFER) = 0 THEN
               DO; /* SIGNAL FOR EOF */
                  CALL ERROR ('EOF MISSING OR COMMENT STARTING IN COLUMN 1.',1);
                  BUFFER = PAD (' /*''/* */ EOF;END;EOF', 80);
               END;
            ELSE CARD_COUNT = CARD_COUNT + 1;  /* USED TO PRINT ON LISTING */
      IF MARGIN_CHOP > 0 THEN
         DO; /* THE MARGIN CONTROL FROM DOLLAR | */
            I = LENGTH(BUFFER) - MARGIN_CHOP;
            REST = SUBSTR(BUFFER, I);
            BUFFER = SUBSTR(BUFFER, 0, I);
         END;
      ELSE REST = '';
      TEXT = BUFFER;
      TEXT_LIMIT = LENGTH(TEXT) - 1;
      IF CONTROL(BYTE('M')) THEN OUTPUT = BUFFER;
      ELSE IF CONTROL(BYTE('L')) THEN
         OUTPUT = I_FORMAT (CARD_COUNT, 4) || ' |' || BUFFER || '|' || REST;
      CP = 0;
   END GET_CARD;

   /*                    THE SCANNER PROCEDURES                    */

CHAR:
   PROCEDURE;
      /* USED FOR STRINGS TO AVOID CARD BOUNDARY PROBLEMS */
      CP = CP + 1;
      IF CP <= TEXT_LIMIT THEN RETURN;
      CALL GET_CARD;
   END CHAR;

SCAN:
   PROCEDURE;
      DECLARE (S1, S2) FIXED;
      CALLCOUNT(3) = CALLCOUNT(3) + 1;
      FAILSOFT = TRUE;
      BCD = '';  NUMBER_VALUE = 0;
   SCAN1:
      DO FOREVER;
         IF CP > TEXT_LIMIT THEN CALL GET_CARD;
         ELSE
            DO; /* DISCARD LAST SCANNED VALUE */
               TEXT_LIMIT = TEXT_LIMIT - CP;
               TEXT = SUBSTR(TEXT, CP);
               CP = 0;
```

```
   END;
/*  BRANCH ON NEXT CHARACTER IN TEXT                  */
DO CASE CHARTYPE(BYTE(TEXT));

   /*  CASE 0  */

   /* ILLEGAL CHARACTERS FALL HERE  */
   CALL ERROR ('ILLEGAL CHARACTER: ' || SUBSTR(TEXT, 0, 1));

   /*  CASE 1  */

   /*  BLANK  */
   DO;
      CP = 1;
      DO WHILE BYTE(TEXT, CP) = BYTE(' ') & CP <= TEXT_LIMIT;
         CP = CP + 1;
      END;
      CP = CP - 1;
   END;

   /*  CASE 2  */

   DO FOREVER;  /* A LETTER:  IDENTIFIERS AND RESERVED WORDS */
      DO CP = CP + 1 TO TEXT_LIMIT;
         IF NOT_LETTER_OR_DIGIT(BYTE(TEXT, CP)) THEN
            DO;  /* END OF IDENTIFIER  */
               IF CP > 0 THEN BCD = BCD || SUBSTR(TEXT, 0, CP);
               S1 = LENGTH(BCD);
               IF S1 > 1 THEN IF S1 <= RESERVED_LIMIT THEN
                  /* CHECK FOR RESERVED WORDS *7
                  DO I = V_INDEX(S1-1) TO V_INDEX(S1) - 1;
                     IF BCD = V(I) THEN
                        DO;
                           TOKEN = I;
                           RETURN;
                        END;
                  END;
               /*  RESERVED WORDS EXIT HIGHER: THEREFORE <IDENTIFIER>*/
               TOKEN = IDENT;
               RETURN;
            END;
      END;
      /*  END OF CARD  */
      BCD = BCD || TEXT;
      CALL GET_CARD;
      CP = -1;
   END;
```

```
/*  CASE 3  */

DO;       /*  DIGIT:  A NUMBER  */
   TOKEN = NUMBER;
   DO FOREVER;
      DO CP = CP TO TEXT_LIMIT;
         S1 = BYTE(TEXT, CP);
         IF S1 < "F0" THEN DO;
            IF PROB_FLAG THEN PROBLEM = PROBLEM || NUMBER_VALUE;
            RETURN;
         END;
         NUMBER_VALUE = 10*NUMBER_VALUE + S1 - "F0";
      END;
      CALL GET_CARD;
   END;
END;

/*  CASE 4  */

DO;       /*  A /:  MAY BE DIVIDE OR START OF COMMENT  */
   CALL CHAR;
   IF BYTE(TEXT, CP) ¬= BYTE('*') THEN
      DO;
         TOKEN = DIVIDE;
         RETURN;
      END;
   /* WE HAVE A COMMENT  */
   S1, S2 = BYTE(' ');
   DO WHILE S1 ¬= BYTE('*') | S2 ¬= BYTE('/');
      IF S1 = BYTE('$') THEN
         DO;   /* A CONTROL CHARACTER  */
            CONTROL(S2) = ¬ CONTROL(S2);
            IF S2 = BYTE('T') THEN CALL TRACE;
            ELSE IF S2 = BYTE('U') THEN CALL UNTRACE;
            ELSE IF S2 = BYTE('|') THEN
               IF CONTROL(S2) THEN
                  MARGIN_CHOP = TEXT_LIMIT - CP + 1;
               ELSE
                  MARGIN_CHOP = 0;
         END;
      S1 = S2;
      CALL CHAR;
      S2 = BYTE(TEXT, CP);
   END;
END;

/*  CASE 5  */
DO:       /*  SPECIAL CHARACTERS  */
```

```
                TOKEN = TX(BYTE(TEXT));
                IF PROB_FLAG THEN PROBLEM = PROBLEM || SUBSTR(TEXT,0,1);
                CP = 1;
                RETURN:
             END;

          END;      /* OF CASE ON CHARTYPE  */
          CP = CP + 1;  /* ADVANCE SCANNER AND RESUME SEARCH FOR TOKEN  */
       END;
    END SCAN;

  /*                          TIME AND DATE                                  */

PRINT_TIME:
    PROCEDURE (MESSAGE, T);
       DECLARE MESSAGE CHARACTER, T FIXED;
       MESSAGE = MESSAGE || T/360000 || ':' || T MOD 360000 / 6000 || ':'
          || T MOD 6000 / 100 || '.';
       T = T MOD 100;   /* DECIMAL FRACTION  */
       IF T < 10 THEN MESSAGE = MESSAGE || '0';
       OUTPUT = MESSAGE || T || '.';
    END PRINT_TIME;

PRINT_DATE_AND_TIME:
    PROCEDURE (MESSAGE, D, T);
       DECLARE MESSAGE CHARACTER, (D, T, YEAR, DAY, M) FIXED;
       DECLARE MONTH(11) CHARACTER INITIAL ('JANUARY', 'FEBRUARY', 'MARCH',
          'APRIL', 'MAY', 'JUNE', 'JULY', 'AUGUST', 'SEPTEMBER', 'OCTOBER',
          'NOVEMBER', 'DECEMBER'),
       DAYS(12) FIXED INITIAL (0, 31, 60, 91, 121, 152, 182, 213, 244, 274,
          305, 335, 366);
       YEAR = D/1000 + 1900;
       DAY = D MOD 1000;
       IF (YEAR & "3") ¬= 0 THEN IF DAY > 59 THEN DAY = DAY + 1; /* ¬ LEAP YEAR*/
       M = 1;
       DO WHILE DAY > DAYS(M);  M = M + 1;  END;
       CALL PRINT_TIME(MESSAGE || MONTH(M-1) || X1 || DAY-DAYS(M-1) ||  ', '
          || YEAR || '.  CLOCK TIME = ', T);
    END PRINT_DATE_AND_TIME;

  /*                          INITIALIZATION                                    */

INITIALIZATION:
    PROCEDURE;
       EJECT_PAGE;
    CALL PRINT_DATE_AND_TIME('    TEACHER''''S ROLE INTERPUTER -- NAVAL ' ||
       'POSTGRADUATE SCHHOL ', DATE_OF_GENERATION, TIME_OF_GENERATION);
       DOUBLE_SPACE;
```

```
CALL PRINT_DATE_AND_TIME ('TODAY IS ', DATE, TIME);
DOUBLE_SPACE;
DO I = 1 TO NT;
   S = V(I);
   IF S = '<NUMBER>' THEN NUMBER = I;  ELSE
   IF S = '<IDENTIFIER>' THEN IDENT = I;  ELSE
   IF S = '/' THEN DIVIDE = I;  ELSE
   IF S = '_|_' THEN EOFILE = I;  ELSE
   IF S = ';' THEN STOPIT(I) = TRUE;  ELSE
   ;
END;
IF IDENT = NT THEN RESERVED_LIMIT = LENGTH(V(NT-1));
ELSE RESERVED_LIMIT = LENGTH(V(NT));
V(EOFILE) = 'EOF';
STOPIT(EOFILE) = TRUE;
CHARTYPE(BYTE(' ')) = 1;
DO I = 0 TO 255;
   NOT_LETTER_OR_DIGIT(I) = TRUE;
END;
DO I = 0 TO LENGTH(ALPHABET) - 1;
   J = BYTE(ALPHABET, I);
   TX(J) = I;
   NOT_LETTER_OR_DIGIT(J) = FALSE;
   CHARTYPE(J) = 2;
END;
DO I = 0 TO 9;
   J = BYTE('0123456789', I);
   NOT_LETTER_OR_DIGIT(J) = FALSE;
   CHARTYPE(J) = 3;
END;
DO I = V_INDEX(0) TO V_INDEX(1) - 1;
   J = BYTE(V(I));
   TX(J) = I;
   CHARTYPE(J) = 5;
END;
CHARTYPE(BYTE('/')) = 4;
/* FIRST SET UP GLOBAL VARIABLES CONTROLLING SCAN, THEN CALL IT */
CP = 0;  TEXT_LIMIT = -1;
TEXT = '';
CONTROL(BYTE('L')) = TRUE;
SY = 0;
FLAG(SY), VAL_FLAG(SY) = FALSE;
VALUE_OF(SY) = 0;
PROB_FLAG = TRUE;
NUMBER_OF(SY) = '';
CALL SCAN;
/* INITIALIZE THE PARSE STACK */
SP = 1;  PARSE_STACK(SP) = EOFILE;
```

```
   END INITIALIZATION;

DUMPIT:
   PROCEDURE;    /* DUMP OUT THE STATISTICS COLLECTED DURING THIS RUN  */
      DOUBLE_SPACE;
      /*  PUT OUT THE ENTRY COUNT FOR IMPORTANT PROCEDURES */
      OUTPUT = 'STACKING DECISIONS= ' || CALLCOUNT(1);
      OUTPUT = 'SCAN               = ' || CALLCOUNT(3);
      OUTPUT = 'FREE STRING AREA   = ' || FREELIMIT - FREEBASE;
   END DUMPIT;

STACK_DUMP:
   PROCEDURE;
      DECLARE LINE CHARACTER;
      LINE = 'PARTIAL PARSE TO THIS POINT IS: ';
      DO I = 2 TO SP;
         IF LENGTH(LINE) > 105 THEN
            DO;
               OUTPUT = LINE;
               LINE = X4;
            END;
         LINE = LINE || X1 || V(PARSE_STACK(I));
      END;
      OUTPUT = LINE;
   END STACK_DUMP;

LOOKUP: PROCEDURE(C);
   DECLARE C CHARACTER, I FIXED;
   DO I = 1 TO SY;
      IF SYMBOL(I) = C THEN RETURN I;
   END;
   RETURN 0;
   END LOOKUP;

ENTER: PROCEDURE(C);
   DECLARE C CHARACTER;
   SY = SY + 1;
   IF SY > MAXSYM THEN DO;
      CALL ERROR(' SYMBOL TABLE OVERFLOW ',2);
      CALL EXIT;
   END;
   SYMBOL(SY) = C;
   NUMBER_OF(SY) = '';
   VALUE_OF(SY) = 0;
   RETURN SY;
   END ENTER;

TRACER: PROCEDURE(C) FIXED;
```

```
    DECLARE C FIXED, L CHARACTER;
    L = ' ';
    DO I1 = MP TO SP;
       L = L||V(PARSE_STACK(I1))||' ';
    END;
    OUTPUT = '('||C||' )'||L;
    END TRACER;

 INDEX: PROCEDURE(C, D) FIXED;
    DECLARE (C, D) CHARACTER, (L, J) FIXED:
    L = LENGTH(D);
    DO J  = 0 TO LENGTH(C) - L;
       IF SUBSTR(C,J ,L) = D THEN RETURN J;
    END;
    RETURN 0;
    END INDEX;

 RANDOM: PROCEDURE(RANGE) FIXED;
    DECLARE RANGE FIXED, RBASE FIXED INITIAL(1),
       RMULT LITERALLY '671555';
    RBASE = RBASE * RMULT;
    RETURN SHR(SHR(RBASE,16) * RANGE,16);
    END RANDOM;

 FIND_VALUE: PROCEDURE(LOW, HIGH, I);
    DECLARE (LOW, HIGH, I, J) FIXED;
    NUMBER_STRING = '';
    DO J = LOW TO HIGH;
       NUMBER_STRING = NUMBER_STRING || J;
    END;
    J = RANDOM(LENGTH(NUMBER_STRING));
    VALUE_OF(I) = BYTE(NUMBER_STRING,J) -."F0";
    END FIND_VALUE;

 GET_VALUE_OF: PROCEDURE;
    DECLARE DEC CHARACTER, (I, J, K, L) FIXED;
    DO I = SY_CNT+1 TO SY;
       IF ¬VAL_FLAG(I) THEN DO;
          IF FLAG(I) THEN DO;
             J = RANDOM(LENGTH(NUMBER_OF(I)));
             VALUE_OF(I) = BYTE(NUMBER_OF(I),J) - "F0";
          END;
          ELSE DO;
             NUMBER_STRING = NUMBER_OF(I);
             L = INDEX(NUMBER_STRING,' ');
             S = SUBSTR(NUMBER_STRING,0,L);
             IF BYTE(S,0) < "F0" THEN DO;
                I1 = LOOKUP(S);
```

```
         K = BYTE(NUMBER_STRING,L+1) - "F0";
         CALL FIND_VALUE(VALUE_OF(I1), K, I);
      END;
      ELSE DO;
         NUMBER_STRING = SUBSTR(NUMBER_STRING,L+1);
         I2 = 0;
         DO J = 0 TO LENGTH(S)-1;
            I2 = I2*10 + BYTE(S,J)-"F0";
         END;
         IF LENGTH(NUMBER_STRING) = 2 THEN DO;
            I1 = LOOKUP(NUMBER_STRING);
            CALL FIND_VALUE(I2, VALUE_OF(I1), I);
         END;
         ELSE DO;
            IF BYTE(NUMBER_STRING,0) = BYTE('-') THEN DO;
               I3 = INDEX(NUMBER_STRING,' ');
               NUMBER_STRING = SUBSTR(NUMBER_STRING,I3+1);
               I1 = LOOKUP(SUBSTR(NUMBER_STRING,0,2));
               I2 = I2 - VALUE_OF(I1);
               I3 = INDEX(NUMBER_STRING,' ');
               I1 = BYTE(NUMBER_STRING,I3+1) - "F0";
               IF I2 > 9 THEN I2 = 9;
               CALL FIND_VALUE(I2, I1, I);
            END;
            ELSE DO;
               L = INDEX(NUMBER_STRING,' ');
               S = SUBSTR(NUMBER_STRING,0,L);
               NUMBER_STRING = SUBSTR(NUMBER_STRING,L+1);
               IF BYTE(S,0) > "F0" THEN DO;
                  I1 = 0;
                  DO J = 0 TO LENGTH(S)-1;
                     I1 = I1*10 + BYTE(S,J)-"F0";
                  END;
                  I3 = LOOKUP(NUMBER_STRING);
                  I1 = I1 - VALUE_OF(I3);
                  CALL FIND_VALUE(I2, I1, I);
               END;
               ELSE DO;
                  L = INDEX(NUMBER_STRING,' ');
                  DEC = SUBSTR(NUMBER_STRING,0,L);
                  IF BYTE(DEC,0) = BYTE('-') THEN DO;
                     DEC = SUBSTR(NUMBER_STRING,2);
                     I1 = LOOKUP(S);
                     I3 = 0;
                     DO J = 0 TO LENGTH(DEC)-1;
                        I3 = I3*10 + BYTE(DEC,J)-"F0";
                     END;
                     I4 = VALUE_OF(I1) - I3;
```

```
                           CALL FIND_VALUE(I2, I4, I);
                        END;
                        ELSE DO;
                           DEC = SUBSTR(NUMBER_STRING,2);
                           I1 = LOOKUP(DEC);
                           I3 = VALUE_OF(I1) - 1;
                           I4 = VALUE_OF(I1) + 1;
                           I1 = 0;
                           DO J = 0 TO LENGTH(DEC)-1;
                              I1 = I1*10 + BYTE(DEC,J)-"FC";
                           END;
                           NUMBER_STRING = '';
                           DO J = I2 TO I3;
                              NUMBER_STRING = NUMBER_STRING || J;
                           END;
                           DO J = I4 TO I1;
                              NUMBER_STRING = NUMBER_STRING || J;
                           END;
                           J = RANDOM(LENGTH(NUMBER_STRING));
                           VALUE_OF(I) = BYTE(NUMBER_STRING,J)-"FC";
                        END;
                     END;
                  END;
               END;
            END;
         END;
      END;
   END;
   END GET_VALUE_OF;

VERT: PROCEDURE;
   DECLARE LINE CHARACTER, LINE_LENGTH FIXED, (I, J, K, L) FIXED;
   OUTPUT = 'VERT';
   DO I = 1 TO 5;
      CALL GET_VALUE_OF;
   DO J = 0 TO SY;
      OUTPUT = SYMBOL(J) || ' VALUE_OF = ' || VALUE_OF(J);
   END;
      LINE = '';
      LINE_LENGTH = 0;
      DO J = 1 TO SY_CNT;
         L = LENGTH(SYMBOL(J));
         LINE_LENGTH = LINE_LENGTH + L;
         I1 = 0;
         IF SYMBOL(J) = 'ANSWER' THEN LINE = LINE || '______';
         ELSE DO;
            DO K = 0 TO L-1 BY 2;
               I1 = I1*10 + VALUE_OF(LOOKUP(SUBSTR(SYMBOL(J),K,2)));
```

```
              END;
          LINE = LINE || I1:
          END;
          IF J < SY_CNT THEN DO WHILE SUBSTR(PROBLEM,LINE_LENGTH,LENGTH(SYMBOL(J+
             1))) ¬= SYMBOL(J+1);
             LINE = LINE || SUBSTR(PROBLEM, LINE_LENGTH, 1);
             LINE_LENGTH = LINE_LENGTH + 1;
          END;
       END;
    DOUBLE_SPACE;
    OUTPUT = LINE;
    DOUBLE_SPACE;
    END;
    END VERT;

HOR: PROCEDURE;
    DECLARE LINE(6) CHARACTER, LINE_LENGTH FIXED, (I, J, K, L) FIXED;
    DECLARE LINE_OUT CHARACTER;
    OUTPUT = 'HOR';
    DO J = 1 TO 6;
       LINE(J) = '';
    END;
    DO I = 1 TO 5;
       CALL GET_VALUE_OF;
    DO J = 0 TO SY;
       OUTPUT = SYMBOL(J) || ' VALUE_OF = ' || VALUE_OF(J);
    END;
       L = 0;
       DO J = 1 TO SY_CNT;
          IF LENGTH(SYMBOL(J)) > L THEN L = LENGTH(SYMBOL(J));
       END;
       L = L/2;
       DO J = 1 TO SY_CNT;
          IF SYMBOL(J) = 'ANSWER' THEN LINE(J) = LINE(J) || '          ' ||
             C_FORMAT('',L): ELSE DO;
             I1 = 0;
             DO K = 0 TO LENGTH(SYMBOL(J))-1 BY 2;
                I1 = I1*10 + VALUE_OF(LOOKUP(SUBSTR(SYMBOL(J),K,2)));
             END;
             LINE(J) = LINE(J) || '          ' || I_FORMAT(I1, L);
             END;
       END;
    END;
    DOUBLE_SPACE;
    DO J = 1 TO SY_CNT;
       OUTPUT = LINE(J);
       IF J = SY_LINE THEN DO;
          LINE_OUT = '';
```

```
          DO K = 1 TO 5;
             LINE_OUT = LINE_OUT || '          ' || SUBSTR('__________',0,L);
          END;
          OUTPUT = LINE_OUT;
       END;
    END;
    DOUBLE_SPACE;
    END HOR;
  /*                    THE SYNTHESIS ALGORITHM FOR XPL                          */

SYNTHESIZE:
PROCEDURE(PRODUCTION_NUMBER);
   DECLARE PRODUCTION_NUMBER FIXED;
   DECLARE COUNT BIT(T), (TEMP, NUM) CHARACTER, DEC FIXED;
   IF CONTROL(BYTE('P')) THEN CALL TRACER(PRODUCTION_NUMBER);

   DO CASE PRODUCTION_NUMBER;
     ;
  /*  <INTERPUTER> ::= <INTERPUTER STATEMENT> ;     */

   DO;
     IF MP ¬= 2 THEN /* WE DIDN'T GET HERE LEGITIMATELY  */
        DO;
          CALL ERROR ('EOF AT INVALID POINT', 1);
          CALL STACK_DUMP;
        END;
     COMPILING = FALSE;
   END;
  /*  <INTERPUTER STATEMENT> ::= <PROBLEM STATEMENT>    */

   DO;
     PROB_FLAG = FALSE;
     SY_CNT = SY;
   END;
  /*  <INTERPUTER STATEMENT> ::= <INTERPUTER STATEMENT> ; <PROBLEM STATEMENT>
        */

   DO;
     PROB_FLAG = FALSE;
     SY_CNT = SY;
   END;
  /*  <INTERPUTER STATEMENT> ::= <INTERPUTER STATEMENT> <RIGHT STATEMENT>    */
```

```
  ;

/*  <INTERPUTER STATEMENT> ::= <INTERPUTER STATEMENT> <DIRECTION>    */

  DO;
     IF VAR(SP) = 'VERT' THEN CALL VERT; ELSE
     IF VAR(SP) =  'HOR' THEN CALL HOR;
     PROB_FLAG = TRUE;
     SY = 0;
     PROBLEM = '';
  END;

/*  <RIGHT STATEMENT> ::= <LEFT PART> <A_NUMBER>    */

  ;

/*  <LEFT PART> ::= <IDENTIFIER =>    */

  DO;
     CALL ENTER(VAR(MP));
     COUNT = TRUE;
     NUMBER_STRING = '';
  END;

/*  <A_NUMBER> ::= <A_NUMBER(> <DIGIT PART>    */

  ;

/*  <A_NUMBER(> ::= A_NUMBER ( <IDENTIFIER>    */

  DO;
     COUNT = FALSE;
     TEMP = VAR(SP);
  END;

/*  <A_NUMBER(> ::= A_NUMBER ( <NUMBER>    */

  DEC = FIXV(SP);

/*  <DIGIT PART> ::= )    */

  IF COUNT THEN DO;
     FLAG(SY) = TRUE;
     NUMBER_OF(SY) = DEC;
     VALUE_OF(SY) = DEC;
     VAL_FLAG(SY) = TRUE;
  END;
  ELSE DO;
```

```
      I1 = LOOKUP(TEMP);
      IF VAL_FLAG(I1) THEN DO;
         VAL_FLAG(SY), FLAG(SY) = TRUE;
         VALUE_OF(SY) = VALUE_OF(I1);
         NUMBER_OF(SY) = NUMBER_OF(I1);
      END;
      ELSE DO;
         VAL_FLAG(SY), FLAG(SY) = FALSE;
         NUMBER_OF(SY) = TEMP;
      END;
   END;

/*  <DIGIT PART> ::= <MINUS IDENTIFIER> <ARROW NUMBER> )     */

   IF COUNT THEN DO;
      I1 = LOOKUP(TEMP);
      IF VAL_FLAG(I1) THEN DO;
         FLAG(SY) = TRUE;
         VAL_FLAG(SY) = FALSE;
         I2 = DEC - VALUE_OF(I1);
         DO I = I2 TO FIXV(SP-1);
            NUMBER_STRING = NUMBER_STRING || I;
         END;
         NUMBER_OF(SY) = NUMBER_STRING;
      END;
      ELSE DO;
         VAL_FLAG(SY), FLAG(SY) = FALSE;
         NUMBER_OF(SY) = DEC || ' ' ||'-' || ' ' || VAR(MP) || ' ' ||FIXV(SP-1);
      END;
   END;

/*  <DIGIT PART> ::= <RIGHT PART> <PLUS NUMBER> <ARROW NUMBER> )     */

   NUMBER_OF(SY) = NUMBER_OF(SY) || ' ' || FIXV(SP-1);

/*  <DIGIT PART> ::= <ARROW NUMBER> <MINUS IDENTIFIER> )     */

   IF CCUNT THEN DO;
      I1 = LOOKUP(VAR(SP-1));
      IF VAL_FLAG(I1) THEN DO;
         I2 = FIXV(MP) - VALUE_OF(I1);
         DO I = DEC TO I2;
            NUMBER_STRING = NUMBER_STRING || I;
         END;
         NUMBER_OF(SY) = NUMBER_STRING;
         FLAG(SY) = TRUE;
         VAL_FLAG(SY) = FALSE;
      END;
```

```
      ELSE DO;
         NUMBER_OF(SY) = DEC || ' ' || FIXV(MP) || ' ' || VAR(SP-1);
         VAL_FLAG(SY), FLAG(SY) = FALSE;
      END;
   END;

/*  <DIGIT PART> ::= <ARROW IDENTIFIER> <MINUS NUMBER> )     */

   IF COUNT THEN DO;
      I1 = LOOKUP(VAR(MP));
      IF VAL_FLAG(I1) THEN DO;
         FLAG(SY) = TRUE;
         VAL_FLAG(SY) = FALSE;
         I2 = VALUE_OF(I1) - FIXV(SP-1);
         DO I = DEC TO I2;
            NUMBER_STRING = NUMBER_STRING || I;
         END;
         NUMBER_OF(SY) = NUMBER_STRING;
      END;
      ELSE DO;
         FLAG(SY), VAL_FLAG(SY) = FALSE;
         NUMBER_OF(SY) = DEC || ' ' || VAR(MP) || ' ' || '-' || ' '||FIXV(SP-1);
      END;
   END;

/*  <DIGIT PART> ::= <ARROW IDENTIFIER> )     */

   IF COUNT THEN DO;
      I1 = LOOKUP(VAR(SP-1));
      IF VAL_FLAG(SY) THEN DO;
         FLAG(SY) = TRUE;
         VAL_FLAG(SY) = FALSE;
         DO I = DEC TO VALUE_OF(I1);
            NUMBER_STRING = NUMBER_STRING || I;
         END;
         NUMBER_OF(SY) = NUMBER_STRING:
      END;
      ELSE DO;
         FLAG(SY), VAL_FLAG(SY) = FALSE;
         NUMBER_OF(SY) = DEC || ' ' || FIXV(SP-1);
      END;
   END;

/*  <DIGIT PART> ::= <ARROW NUMBER> )     */

   IF COUNT THEN DO;
      FLAG(SY) = TRUE;
      DO I = DEC TO FIXV(MP);
```

```
            NUMBER_STRING = NUMBER_STRING || I:
         END;
         NUMBER_OF(SY) = NUMBER_STRING;
         VAL_FLAG(SY) = FALSE;
      END;
      ELSE DO;
         I1 = LOOKUP(TEMP);
         IF VAL_FLAG(I1) THEN DO;
            FLAG(SY) = TRUE;
            DO I = VALUE_OF(I1) TO FIXV(MP);
               NUMBER_STRING = NUMBER_STRING || I;
            END;
            NUMBER_OF(SY) = NUMBER_STRING;
            VAL_FLAG(SY) = FALSE;
         END;
         ELSE DO;
            VAL_FLAG(SY), FLAG(SY) = FALSE;
            NUMBER_OF(SY) = TEMP || ' ' || FIXV(MP);
         END;
      END;

   /*  <ARROW NUMBER> ::= - > <NUMBER>     */

      FIXV(MP) = FIXV(SP);

   /*  <ARROW IDENTIFIER> ::= - > <IDENTIFIER>     */

      VAR(MP) = VAR(SP);

   /*  <RIGHT PART> ::= <ARROW IDENTIFIER> <MINUS NUMBER> , <IDENTIFIER>     */

      IF COUNT THEN NUMBER_OF(SY) = DEC || ' ' || VAR(MP); ELSE
      NUMBER_OF(SY) = TEMP || ' ' || VAR(MP):

   /*  <PLUS NUMBER> ::= + <NUMBER>     */

      FIXV(MP) = FIXV(SP);

   /*  <MINUS NUMBER> ::= - <NUMBER>     */

      FIXV(MP) = FIXV(SP);

   /*  <MINUS IDENTIFIER> ::= - <IDENTIFIER>     */

      VAR(MP) = VAR(SP);

   /*  <IDENTIFIER => ::= <IDENTIFIER> =     */
```

```
  ;
/*  <PROBLEM STATEMENT> ::= <EXPRESSION> <RELATION> <EXPRESSION>    */
  ;
/*  <EXPRESSION> ::= <TERM>    */
  ;
/*  <EXPPESSION> ::= <EXPPESSION> <+> <TERM>    */
  ;
/*  <EXPRESSION> ::= <EXPRESSION> <-> <TERM>    */
  ;
/*  <TERM> ::= <PRIMARY>    */
  ;
/*  <TERM> ::= <TERM> * <PRIMARY>    */
  ;
/*  <TERM> ::= <TERM> / <PRIMARY>    */
  ;
/*  <PRIMARY> ::= <NUMBER>    */
  ;
/*  <PRIMARY> ::= <IDENTIFIER>    */
  CALL ENTER(VAR(MP));
/*  <PRIMARY> ::= <(2> <EXPRESSION> )    */
  ;
/*  <PRIMARY> ::= ANSWER    */
  CALL ENTER(VAR(MP));
/*  <(2> ::= (    */
  ;
```

```
/*  <-> ::= -      */
  ;
/*  <+> ::= +      */
  ;
/*  <RELATION> ::= =     */
  IF PROB_FLAG THEN SY_LINE = SY;
/*  <RELATION> ::= <     */
  IF PROB_FLAG THEN SY_LINE = SY;
/*  <RELATION> ::= >     */
  IF PROB_FLAG THEN SY_LINE = SY;
/*  <RELATION> ::= ¬ =     */
  IF PROB_FLAG THEN SY_LINE = SY;
/*  <RELATION> ::= ¬ <     */
  IF PROB_FLAG THEN SY_LINE = SY;
/*  <RELATION> ::= ¬ >     */
  IF PROB_FLAG THEN SY_LINE = SY;
/*  <RELATION> ::= < =     */
  IF PROB_FLAG THEN SY_LINE = SY;
/*  <RELATION> ::= > =     */
  IF PROB_FLAG THEN SY_LINE = SY;
/*  <RELATION> ::= ?     */
  IF PROB_FLAG THEN SY_LINE = SY;
/*  <DIRECTION> ::= HOR     */
```

```
    ;
 /*  <DIRECTION> ::= VERT     */

    ;

    END;
END SYNTHESIZE;

  /*                SYNTACTIC PARSING FUNCTIONS                              */

RIGHT_CONFLICT:
   PROCEDURE (LEFT) BIT(1);
      DECLARE LEFT FIXED;
      /*  THIS PROCEDURE IS TRUE IF TOKEN IS ¬ A LEGAL RIGHT CONTEXT OF LEFT*/
      RETURN ("C0" & SHL(BYTE(C1(LEFT), SHR(TOKEN,2)), SHL(TOKEN,1)
         & "C6")) = 0;
   END RIGHT_CONFLICT;


RECOVER:
   PROCEDURE;
      /* IF THIS IS THE SECOND SUCCESSIVE CALL TO RECOVER, DISCARD ONE SYMBOL */
      IF ¬ FAILSOFT THEN CALL SCAN;
      FAILSOFT = FALSE;
      DO WHILE ¬ STOPIT(TOKEN);
         CALL SCAN;  /* TO FIND SOMETHING SOLID IN THE TEXT  */
      END;
      DO WHILE RIGHT_CONFLICT (PARSE_STACK(SP));
         IF SP > 2 THEN SP = SP - 1;  /* AND IN THE STACK   */
         ELSE CALL SCAN;  /* BUT DON'T GO TOO FAR  */
      END;
      OUTPUT = 'RESUME:' || SUBSTR(POINTER, TEXT_LIMIT-CP+MARGIN_CHOP+7);
   END RECOVER;

STACKING:
   PROCEDURE BIT(1);  /* STACKING DECISION FUNCTION */
      CALLCOUNT(1) = CALLCOUNT(1) + 1;
      DO FOREVER:     /* UNTIL RETURN  */
         DO CASE SHR(BYTE(C1(PARSE_STACK(SP)),SHR(TOKEN,2)),SHL(3-TOKEN,1)&6)&3;

            /*  CASE 0   */
            DO;    /* ILLEGAL SYMBOL PAIR  */
               CALL ERROR('ILLEGAL SYMBOL PAIR: ' || V(PARSE_STACK(SP)) || X1 ||
                  V(TOKEN), 1);
               CALL STACK_DUMP;
               CALL RECOVER;
            END;
```

```
            /*  CASE 1  */

            RETURN TRUE;      /*  STACK TOKEN  */

            /*  CASE 2  */

            RETURN FALSE;     /* DON'T STACK IT YET  */

            /*  CASE 3  */

            DO;       /* MUST CHECK TRIPLES  */
               J = SHL(PARSE_STACK(SP-1), 16) + SHL(PARSE_STACK(SP), 8) + TOKEN;
               I = -1;  K = NC1TRIPLES + 1;  /* BINARY SEARCH OF TRIPLES  */
               DO WHILE I + 1 < K;
                  L = SHR(I+K, 1);
                  IF C1TRIPLES(L) > J THEN K = L;
                  ELSE IF C1TRIPLES(L) < J THEN I = L;
                  ELSE RETURN TRUE;  /* IT IS A VALID TRIPLE  */
               END;
               RETURN FALSE;
            END;

         END;    /* OF DO CASE  */
      END;   /*  OF DO FOREVER */
   END STACKING;

PR_OK:
   PROCEDURE(PRD) BIT(1);
      /* DECISION PROCEDURE FOR CONTEXT CHECK OF EQUAL OR IMBEDDED RIGHT PARTS*/
      DECLARE (H, I, J, PRD) FIXED;
      DO CASE CONTEXT_CASE(PRD);

         /*  CASE 0 -- NO CHECK REQUIRED  */

         RETURN TRUE;

         /*  CASE 1 -- RIGHT CONTEXT CHECK  */

         RETURN ¬ RIGHT_CONFLICT (HDTB(PRD));

         /*  CASE 2 -- LEFT CONTEXT CHECK  */

         DO;
            H = HDTB(PRD) - NT;
            I = PARSE_STACK(SP - PRLENGTH(PRD));
            DO J = LEFT_INDEX(H-1) TO LEFT_INDEX(H) - 1;
               IF LEFT_CONTEXT(J) = I THEN RETURN TRUE;
```

```
            END;
            RETURN FALSE;
         END;

         /*  CASE 3 -- CHECK TRIPLES  */

         DO;
            H = HDTB(PRD) - NT;
            I = SHL(PARSE_STACK(SP - PRLENGTH(PRD)), 8) + TOKEN;
            DO J = TRIPLE_INDEX(H-1) TO TRIPLE_INDEX(H) - 1;
               IF CONTEXT_TRIPLE(J) = I THEN RETURN TRUE;
            END;
            RETURN FALSE;
         END;

      END:  /* OF DO CASE  */
   END PR_OK:

  /*                          ANALYSIS ALGORITHM                                  */

REDUCE:
   PROCEDURE;
      DECLARE (I, J, PRD) FIXED;
      /* PACK STACK TOP INTO ONE WORD */
      DO I = SP - 4 TO SP - 1;
         J = SHL(J, 8) + PARSE_STACK(I);
      END;
      DO PPD = PR_INDEX(PARSE_STACK(SP)-1) TO PR_INDEX(PARSE_STACK(SP)) - 1;
         IF (PRMASK(PRLENGTH(PRD)) & J) = PRTB(PRD) THEN
            IF PR_OK(PRD) THEN
            DO;  7* AN ALLOWED REDUCTION */
               MP = SP - PRLENGTH(PRD) + 1; MPP1 = MP + 1;
               CALL SYNTHESIZE(PRDTB(PRD));
               SP = MP:
               PARSE_STACK(SP) = HDTB(PRD);
               RETURN;
            END;
      END;
      /* LOOK UP HAS FAILED, ERROR CONDITION */
      CALL ERROR('NO PRODUCTION IS APPLICABLE',1);
      CALL STACK_DUMP;
      FAILSOFT = FALSE;
      CALL RECOVER;
   END REDUCE;

COMPILATION_LOOP:
   PROCEDURE:
      COMPILING = TRUE;
```

```
      DO WHILE COMPILING;     /* ONCE AROUND FOR EACH PRODUCTION (REDUCTION)  */
         DO WHILE STACKING;
            SP = SP + 1;
            IF SP = STACKSIZE THEN
               DO;
                  CALL ERROR ('STACK OVERFLOW *** CHECKING ABORTED ***', 2);
                  RETURN;   /* THUS ABORTING CHECKING */
               END;
            PARSE_STACK(SP) = TOKEN;
            VAR(SP) = BCD;
            IF PROB_FLAG THEN PROBLEM = PROBLEM || VAR(SP);
            FIXV(SP) = NUMBER_VALUE;
            CALL SCAN;
         END;

         CALL REDUCE;
      END;     /* OF DO WHILE COMPILING  */
   END COMPILATION_LOOP;

PRINT_SUMMARY:
   PROCEDURE;
      DECLARE I FIXED;
      CALL PRINT_DATE_AND_TIME ('END OF CHECKING ', DATE, TIME);
      OUTPUT = '';
      OUTPUT = CARD_COUNT || ' CARDS WERE CHECKED.';
      IF ERROR_COUNT = 0 THEN OUTPUT = 'NO ERRORS WERE DETECTED.';
      ELSE IF ERROR_COUNT > 1 THEN
         OUTPUT = ERROR_COUNT || ' ERRORS (' || SEVERE_ERRORS
            || ' SEVERE) WERE DETECTED.';
      ELSE IF SEVERE_ERRORS = 1 THEN OUTPUT = 'ONE SEVERE ERROR WAS DETECTED.';
         ELSE OUTPUT = 'ONE ERROR WAS DETECTED.';
      IF PREVIOUS_ERROR > 0 THEN
         OUTPUT = 'THE LAST DETECTED ERROR WAS ON LINE ' || PREVIOUS_ERROR
            || PERIOD;
      IF CONTROL(BYTE('D')) THEN CALL DUMPIT;
      DOUBLE_SPACE;
      CLOCK(3) = TIME;
      DO I = 1 TO 3;   /* WATCH OUT FOR MIDNIGHT */
         IF CLOCK(I) < CLOCK(I-1) THEN CLOCK(I) = CLOCK(I) +  8640000;
      END;
      CALL PRINT_TIME ('TOTAL TIME IN CHECKER    ', CLOCK(3) - CLOCK(0));
      CALL PRINT_TIME ('SET UP TIME              ', CLOCK(1) - CLOCK(0));
      CALL PRINT_TIME ('ACTUAL CHECKING TIME     ', CLOCK(2) - CLOCK(1));
      CALL PRINT_TIME ('CLEAN-UP TIME AT END     ', CLOCK(3) - CLOCK(2));
      IF CLOCK(2) > CLOCK(1) THEN   /* WATCH OUT FOR CLOCK BEING OFF */
      OUTPUT = 'CHECKING RATE: ' || 6000*CARD_COUNT/(CLOCK(2)-CLOCK(1))
         || ' CARDS PER MINUTE.';
   END PRINT_SUMMARY;
```

```
MAIN_PROCEDURE:
   PROCEDURE;
      CLOCK(0) = TIME;   /* KEEP TRACK OF TIME IN EXECUTION */
      CALL INITIALIZATION;

      CLOCK(1) = TIME;

      CALL COMPILATION_LOOP;

      CLOCK(2) = TIME;

      /* CLOCK(3) GETS SET IN PRINT_SUMMARY */
      CALL PRINT_SUMMARY;

   END MAIN_PROCEDURE;

CALL MAIN_PROCEDURE;
RETURN SEVERE_ERRORS;

EOF EOF EOF
```

BIBLIOGRAPHY

1. Atkinson, R.C. and Wilson, H.A., Computer-Assisted Instruction, A Book of Readings, Academic Press, 1969.

2. Bryan, G.L., "Computers and Education," Computers and Automation, v. 18, pp. 16-19, March 1969.

3. Buchnall, D.D. and Allen D.W., The Computer in American Education, John Wiley and Sons, Inc., 1962.

4. Coulson, J.E., Programmed Learning and Computer-Based Instruction, John Wiley and Sons, Inc., 1962.

5. Dunca, E.R., and others, Modern School Mathematics Structure and Use, California State Department of Education, 1970.

6. Engvold, K.J. and Hughes, J.L., "A Model for a Multifunctional Teaching System," Communication of ACM, v. 10, pp. 339-342, June 1967.

7. Entelek, Incorporated, Project Number 154-245, Computer-Assisted Instruction, A Survey of the Literature, by A.E. Hickey and J.M. Newton, January 1967.

8. Ferguson, R.L., "Computer Assistance for Individualizing Instruction," Computers and Automation, v. 19, pp. 27-29, March 1970.

9. Gerald, R.W., Computers and Education, McGraw-Hill, 1967.

10. Hoffman, R.B. and Seagle, J.P., "A Problem Oriented Computer-Based Instructional Procedure," 24th Conference Proceedings of the ACM, pp. 97-110, 1969.

11. Jackson, P.H., The Teacher and the Machine, University of Pittsburgh Press, 1968.

12. Kerr, E.G., Ting, T.C. and Walden, W.E., "A Control Program for Computer Assisted Instruction on a General Purpose Computer," 24th Conference Proceedings of the ACM, pp. 111-116, 1969.

13. Long, H.S. and Schwartz, H.A., "Instruction by Computer," Datamation, v. 12, pp. 73-87, September 1966.

14. McKeeman, W.H., Horning, J.J. and Wortman, D.B., A Compiler Generator, Prentice-Hall, 1970.

15. Oettinger, A.C., Run, Computer, Run, Harvard Press, 1969.

16. Skinner, B.F., The Technology of Teaching, Appleton-Century-Crofts, 1968.

17. Suppes, P., "The Use of Computers in Education," Scientific American, v. 215, pp. 207-220, September 1966.

18. Suppes, P., "Computer Technology and the Future of Education," Phi Delta Kappan, April 1968.

19. Uhr, L., "Teaching Machine Programs that Generate Problems as a Function of Interaction with Students," 24th Conference Proceedings of the ACM, pp. 125-134, 1969.

20. Zinn, K.L., "Instructional Uses of Interactive Computer Systems," Datamation, v. 14, pp. 22-27, September 1968.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	LTjg R.C. Bolles, USN Department of Mathematics, Code 53 Bq Naval Postgraduate School Monterey, California 93940	1
4.	LTjg Allen Roberts, USN Department of Mathematics, Code 53 Ro Naval Postgraduate School Monterey, California 93940	1
5.	LT Ronald J. Wools, USN 8224 Royal Gorge Drive San Diego, California 92119	1

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY *(Corporate author)* Naval Postgraduate School Monterey, California 93940	2a. REPORT SECURITY CLASSIFICATION Unclassified
	2b. GROUP
3 REPORT TITLE A General Problem Describer for Computer Assisted Instruction	
4 DESCRIPTIVE NOTES *(Type of report and, inclusive dates)* Master's Thesis; June 1971	
5 AUTHOR(S) *(First name, middle initial, last name)* Ronald J. Wools	

6 REPORT DATE June 1971	7a. TOTAL NO. OF PAGES 99	7b. NO. OF REFS 20
8a. CONTRACT OR GRANT NO	9a. ORIGINATOR'S REPORT NUMBER(S)	
b. PROJECT NO.		
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*	
d.		

10 DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

Currently in computer assisted instruction systems a number of problems are presented to each student during a problem session and each individual problem is specified by the author of the session. A better approach might be to provide the author with a language in which he can describe the general type of problem he wants his students to be taught and let the machine generate the specific problems. This would relieve the teacher of the task of writing out a whole series of problems for each general concept he wishes to teach. This thesis presents a subset of English and mathematical notation which the teacher can use to describe a general problem type. The PROBLEM DESCRIPTION PROCESSOR accepts the general problem description and produces a low level language which is used by the PROBLEM DESCRIPTION INTERPRETER to produce specific problems. This system works for fourth grade arithmetic problems and could be extended for use in other areas of instruction.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

Security Classification

A-314

Security Classification

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computer-Assisted Instruction						
Computer-Aided Learning						
Computer-Based Instruction						
Learning						
Problem Describer						
Teaching						
Tutoring						

DD FORM 1 NOV 65 1473 (BACK)
N 0101-807-6821

Security Classification